

NO-A184 895

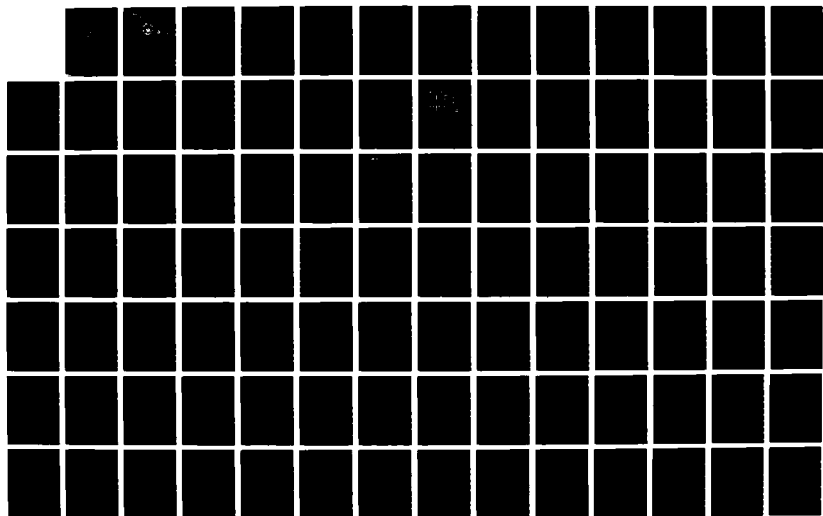
IMPLEMENTATION OF AN FIR BAND PASS FILTER USING A  
BIT-SLICE PROCESSOR(U) NAVAL POSTGRADUATE SCHOOL  
MONTEREY CA D W PURDY JUN 87

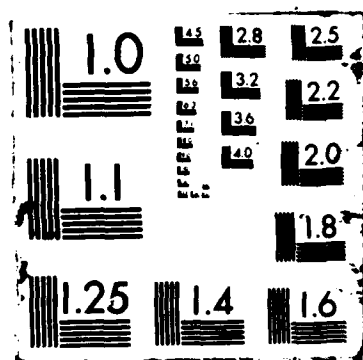
1/2

UNCLASSIFIED

F/G 12/6

NL





AD-A184 895

# NAVAL POSTGRADUATE SCHOOL

Monterey, California

2

DTIC FILE COPY



DTIC  
ELECTED  
OCT 07 1987  
S D

## THESIS

IMPLEMENTATION OF AN FIR BAND PASS FILTER  
USING A BIT-SLICE PROCESSOR

by

Darrel Wayne Purdy

June 1987

Thesis Advisor:

Chin-Hwa Lee

Approved for public release; distribution is unlimited

87 9 25 130

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

## REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b RESTRICTIVE MARKINGS		
2a SECURITY CLASSIFICATION AUTHORITY			3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited		
2b DECLASSIFICATION/DOWNGRADING SCHEDULE					
4 PERFORMING ORGANIZATION REPORT NUMBER(S)			5 MONITORING ORGANIZATION REPORT NUMBER(S)		
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School		6b OFFICE SYMBOL (If applicable) Code 62		7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
6c ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000			7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000		
8a NAME OF FUNDING/SPONSORING ORGANIZATION		8b OFFICE SYMBOL (If applicable)		9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c ADDRESS (City, State, and ZIP Code)		10 SOURCE OF FUNDING NUMBERS			
		PROGRAM ELEMENT NO		PROJECT NO	TASK NO
				WORK UNIT ACCESSION NO	
11 TITLE (Include Security Classification) IMPLEMENTATION OF AN FIR BAND PASS FILTER USING A BIT-SLICE PROCESSOR					
12 PERSONAL AUTHOR(S) Purdy, Darrel, W.					
13a TYPE OF REPORT Master's Thesis		13b TIME COVERED FROM _____ TO _____		14 DATE OF REPORT (Year, Month, Day) 1987, June	
15 PAGE COUNT 164					
16 SUPPLEMENTARY NOTATION					
17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	FIR Band Pass Filter; Microprocessor; Bit-Slice		
19 ABSTRACT (Continue on reverse if necessary and identify by block number) A 13th order FIR filter for digital image processing is implemented in microcode using the Am29203 bit-slice evaluation board of ADVANCED MICRO DEVICES. To meet this requirement, the filter is first implemented in Fortran. Then the results of both implementations are used for timing comparisons. Although non-optimal bit-slice devices are used on the evaluation board, a time of 11 microseconds is achieved, as compared to the 100 microseconds achieved in the Fortran implementation. Theoretical estimates of 2.65 microseconds and 0.78 microseconds are obtained for high speed Am2900 bit-slice devices and VITESSE's Gallium Arsenide bit-slice devices respectively. It is shown that, although the initial learning period for bit-slice devices is high, once learned, a skillful bit-slice designer can implement a simple filter design in minimal time with significant results in time savings.					
20 DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS			21 ABSTRACT SECURITY CLASSIFICATION Unclassified		
22a NAME OF RESPONSIBLE INDIVIDUAL Prof. Chin-Hwa Lee			22b TELEPHONE (Include Area Code) (408) 646-2190		22c OFFICE SYMBOL Code 62Le

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

#19 - ABSTRACT - (CONTINUED)

A brief discussion of bit-slice techniques is presented and an argument is proposed as to whether the bit-slice is a methodology or a device. The most recent introduction of Gallium Arsenide devices is included in the discussion.

In addition to the implementation of the filter, its characteristics as well as its equation representations are presented. A discussion about noise and quantization effects using this digital filter is also presented.

Finally, two appendices are included. The first appendix presents the use of the commercial software SMARTCOM II with the IBM PC to emulate the user terminal for the monitor system of the Am29203 evaluation board. The second appendix presents a detailed look at the bit-slice microcode used to implement the filter.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

Approved for public release; distribution is unlimited

Implementation of an FIR Band Pass Filter  
Using a Bit-Slice Processor

by

Darrel Wayne Purdy  
Lieutenant, United States Navy  
B.S.E.E., University of Oklahoma, May 1980

Submitted in partial fulfillment of the  
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

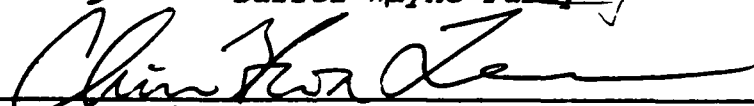
from the

NAVAL POSTGRADUATE SCHOOL  
June 1987

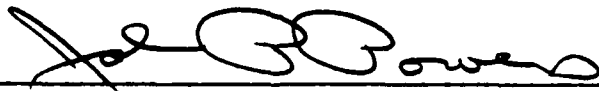
Author:


  
Darrel Wayne Purdy

Approved by:

  
Chin-Hwa Lee, Thesis Advisor

  
Mitchell L. Cotton, Second Reader

  
John P. Powers, Chairman  
Department of Electrical and  
Computer Engineering

  
Gordon E. Schacher  
Dean of Science and Engineering

## ABSTRACT

A 13th order FIR filter for digital image processing is implemented in microcode using the Am29203 bit-slice evaluation board of ADVANCED MICRO DEVICES. To meet this requirement, the filter is first implemented in Fortran. Then the results of both implementations are used for timing comparisons. Although non-optimal bit-slice devices are used on the evaluation board, a time of 11 microseconds is achieved, as compared to the 100 microseconds achieved in the Fortran implementation. Theoretical estimates of 2.65 microseconds and 0.78 microseconds are obtained for high speed Am2900 bit-slice devices and VITESSE's Gallium Arsenide bit-slice devices respectively. It is shown that, although the initial learning period for bit-slice devices is high, once learned, a skillful bit-slice designer can implement a simple filter design in minimal time with significant results in time savings. + /

A brief discussion of bit-slice techniques is presented and an argument is proposed as to whether the bit-slice is a methodology or a device. The most recent commercial introduction of Gallium Arsenide devices is included in the discussion.

In addition to the implementation of the filter, its characteristics as well as its equation representations are

Finally, two appendices are included. The first appendix presents the use of the commercial software SMARTCOM II with the IBM PC to emulate the user terminal for the monitor system of the Am29203 evaluation board. The second appendix presents a detailed look at the bit-slice microcode used to implement the filter.

[illegible]



## TABLE OF CONTENTS

I.	INTRODUCTION -----	9
	A. GENERAL BACKGROUND -----	9
	B. METHOD OF IMPLEMENTATION DEVELOPMENT -----	11
	C. BENEFIT OF STUDY -----	12
II.	BIT-SLICE METHODOLOGY -----	14
	A. INTRODUCTION -----	14
	B. BIT-SLICE HISTORY AND BASIC CONCEPT -----	14
	C. SIMPLE PROCESSOR USING BASIC BIT-SLICE COMPONENTS -----	18
	D. TYPICAL MACRO AND MICRO INSTRUCTIONS -----	28
	E. BIT-SLICE: METHODOLOGY OR DEVICE -----	35
III.	FORTRAN IMPLEMENTATION OF FIR FILTER -----	39
	A. INTRODUCTION OF FIR DIGITAL FILTER -----	39
	B. "DSL" PROGRAM IMPLEMENTATION -----	40
	C. FORTRAN IMPLEMENTATION -----	50
	D. FIXED POINT IMPLEMENTATION AND QUANTIZATION NOISE EFFECTS -----	55
IV.	BIT-SLICE IMPLEMENTATION -----	71
	A. INTRODUCTION -----	71
	B. USE OF EVALUATION BOARD COMPONENTS -----	74
	C. BIT-SLICE IMPLEMENTATION OF THE FIR FILTER -----	77
	D. FORTRAN AND BIT-SLICE IMPLEMENTATION SPEED COMPARISONS -----	88
V.	CONCLUSIONS -----	97

APPENDIX A: TERMINAL EMULATION USING SMARTCOM II -----	100
APPENDIX B: DOCUMENTATION FOR MICROROUTINES -----	107
APPENDIX C: FORTRAN PROGRAM OF FIR FILTER WITH CPU TIMING ROUTINE ADDED -----	158
LIST OF REFERENCES -----	160
BIBLIOGRAPHY -----	162
INITIAL DISTRIBUTION LIST -----	163

### ACKNOWLEDGEMENTS

I wish to gratefully acknowledge by thesis advisor, Professor Chin-Hwa Lee, who provided assistance and insight in the completion of this thesis.

I would also like to express my gratitude to Professor Mitchell L. Cotton for his time and input.

Further, I would like to thank the Defense Mapping Agency System Center for sponsoring the use of the Am29203 Evaluation Board and to ADVANCED MICRO DEVICES (AMD) for providing an extra copy of the Am29203 evaluation board user's guide.

Finally, I would like to express my appreciation to my entire family who supported me during this period and especially my wife and children without whose loving patience I could not have completed this thesis.

## I. INTRODUCTION

### A. GENERAL BACKGROUND

The bit-slice method of computer processor organization originated in the 1970's as an efficient partitioning of the arithmetic and logic unit (ALU) circuitry into convenient LSI components. These components (the "bit-slices") are then applied in a parallel data-path organization to construct processors having any desired data-path width (constrained of course to be a multiple of the basic "bit-slice" size). Since the introduction of bit-slice components, variations and extensions of the original methodology have appeared. Generally the methods involved reflect the following characteristics:

- 1) circuit technology reflecting an emphasis of speed (e.g., bipolar or the most recent introduction of Gallium Arsenide devices [Ref. 1]) rather than density (e.g., conventional MOS microprocessors),
- 2) use of microprogramming to implement either standard or custom instructions (usually facilitated by a separate, replaceable ROM control store), and
- 3) related to 2) above, capability of realizing variable instruction set computers.

As the variety and scope of applications of bit-slice devices has evolved, it has become common to refer to the related methodology as simply "bit-slice". Therefore, in this thesis, wherever reference is made to the unqualified

term bit-slice, it is this general methodology which is referred to.

Of interest in military applications is the use of bit-slice in the redesign of older equipment to emulate existing instruction sets while increasing speed and reliability. Generally, however, the main use of bit-slice is for speed and it has emerged as the dominant technology in high-performance graphics. Because of the complexity of bit-slice microprogramming, much time is necessarily spent toward researching and developing the skills needed in implementing algorithms using this approach. Chapter II introduces the method of bit-slice and its primary components and additionally offers some examples of the recent advances made in this area.

The main thrust of this study was to implement an image processing FIR filter using the methodology of bit-slice. Image processing has a wide range of military applications and the filters used in image processing are just a small part of a very broad area of research. The filter, as presented thoroughly in Chapter III, is a color band pass filter having a carrier frequency of 3.58 MHz and is defined as follows [Ref. 2]:

$$H(Z) = (1-Z^{-1})^2(1-Z^{-2})^2(1+Z^{-3})(1+Z^{-4})$$

The primary goal of course was to minimize the time used to run this filter through standard and bit-slice methods.

Chapter III presents a standard approach using Fortran programming. Necessarily, a secondary emphasis was placed on investigating the advantages of using FIR filters and a special emphasis was placed on the quantization effects produced using these digital filters.

For the bit-slice implementation, the AM29203 evaluation board will be used. This tool allows the user to develop and analyze microprograms through the use of a monitor using a screen-oriented terminal. A description of this tool as well as the implementation of the FIR filter using it is presented in Chapter IV. The AM29203 evaluation board posed some limitations due to the fact that high speed was not a design objective of the evaluation board. The onboard memory is slow and the available look-ahead carry generator for the ALU was not used. However, the theoretical speed which can be achieved is presented along with the actual speed achieved and is compared to that of the Fortran implementation. Finally, the conclusions of this study are presented in Chapter V.

#### B. METHOD OF IMPLEMENTATION DEVELOPMENT

Again, the primary goal was to minimize the time used by the filter using the bit-slice implementation. The proposed method for achieving this goal was as follows:

- 1) Implement the filter in floating point using Fortran programming methods.
- 2) Emulate implementation of the filter in fixed point using Fortran programming methods.

- 3) Implement the filter using 68000 assembly language.
- 4) Implement the filter using bit-slice methods.

The third step, although looked at, was found to be unnecessary. However, if additional time had been available, it would have given a more interesting comparison between the speed of the bit-slice implementation as compared to other methods. Using the method of approach as stated above, a better understanding of the algorithm was achieved, a logical progression of development occurred, and comparisons in speed of implementation between Fortran and bit-slice methods then became available.

#### C. BENEFIT OF STUDY

This study proved to be of great personal benefit in bringing together and solidifying many areas of study learned while at the Naval Postgraduate School. A better understanding was achieved in the areas of filter design and its associated algorithms and limitations; Fortran, assembly and micro level programming and their interrelationships were better understood; and finally, a better understanding was achieved in the application of commercially available hardware and software. This personal benefit will hopefully result in some applied benefit to the Navy.

For Dr. C. H. Lee's interests in this area of image processing, this study achieved two primary goals. First, the FIR filter was successfully implemented using bit-slice methodology. Secondly, the Am29203 evaluation board was

successfully interfaced with an IBM personal computer to allow for the creating and storing of files and for the easy transfer of large amounts of data from stored files to the evaluation board. This last item is documented in Chapter IV and Appendix A.



## II. BIT-SLICE METHODOLOGY

### A. INTRODUCTION

It has been shown that the bit-slice approach, using the simplest bit-serial processor, provides the maximum computational power. [Ref. 3] Commercially, however, when we speak of bit-slice, we are generally referring to 4-bit slice processors such as those offered by ADVANCED MICRO DEVICES (AMD). In this chapter, the bit-slice methodology will be discussed and an example will be given using basic bit-slice components to build a simple microprocessor. Then a typical macro and micro instruction will be introduced using this simple microprocessor. Finally, an argument as to whether bit-slice is a methodology or a device will be presented and discussed.

### B. BIT-SLICE HISTORY AND BASIC CONCEPT

In 1974, Monolithic Memories Inc. introduced the first bit-slice device, marketed as a microcontroller. Several other companies joined in making bit-slice microprocessor devices and by 1978, six companies were offering families of devices classified as bit-slice microprogrammable processor sets. Of these six, all were 4 bit-slice families with the one exception of Intel which offered an unsuccessful 2-bit family. [Ref. 4] During this period, AMD emerged as the leader in bit-slice technology mainly due the design support

the manufacturer offered by way of data sheets and application notes. Because of the critical need for this type of support in designing with bit-slice components due to its design complexity, it is apparent why AMD bit-slice emerged as and is still considered to be the standard of bit-slice technology. Because of this standard, any further references in this paper to bit-slice technology will assume to mean the 4 bit-slice as offered by AMD unless otherwise noted.

Two important concepts must be understood concerning bit-slice methods. The basic underlying concept is that in bit-slice, the data flow is sliced vertically into 4-bit CPU slices and these slices are then joined together horizontally to form microprocessors in increments of 4 bits. In the example which will be presented later in this section, four 4 bit-slices are joined together to form a 16 bit microprocessor. Secondly, the bit-slice technology is most generally hidden from the end user. This is because bit-slice is a method for microprogramming machine-level instructions or macro instructions. As shown in Figure 2.1, levels A and B, the end user would normally be concerned with the basic source code or at most, the assembly source code of a computer. These codes would then be run through a compiler or assembler program (software) to generate machine level instructions. Figure 2.1, level C, then illustrates how these machine-level instructions (software)

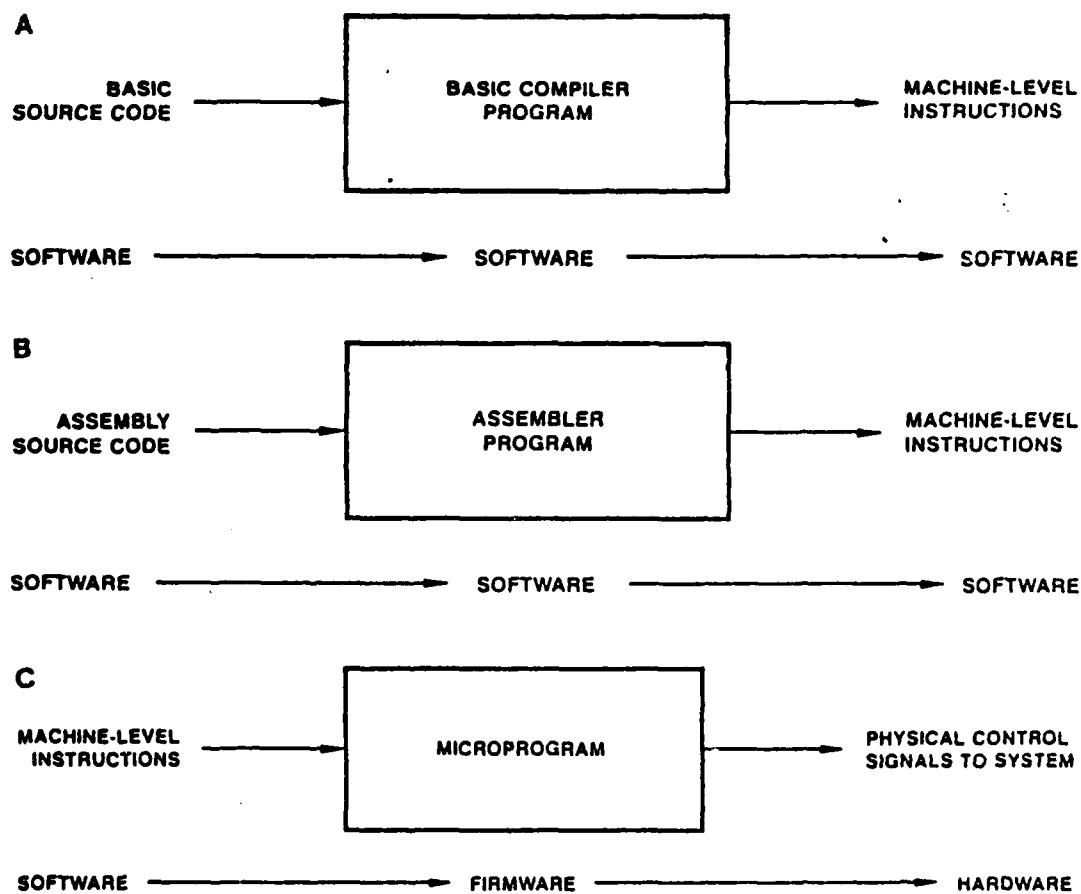


Figure 2.1 Instruction Levels [Ref. 5]

are microprogrammed (firmware) to enable physical control signals to the system (hardware). Therefore, the bit-slice design can be microprogrammed to support any instruction set through the use of hardware and firmware. A good example of how bit-slice is hidden from the end user was the introduction in 1980 by Univac of its model 1100/60 computer using bit-slice microprocessors in the central processing unit. Despite the major change at the microprogramming level, the outward appearance and instruction set was the same as the previous 1100 series. [Ref. 6]

In bit-slice architecture, most of the architecture is left to the user's definitions through the use of interconnections and the microprogram. The advantages offered with bit-slice design are fast complex design capabilities relative to hardware, documentation is forced, and upgrades are made easily by simply replacing PROMs. Bit slice methods are typically used for machines with long words, machines with special instruction sets, and with high machine speeds. These last two categories make the bit-slice particularly well suited for military application, especially in the redesigning or upgrading of older equipment. Also, because of its speed capabilities, the bit-slice processor has emerged as the dominant technology in high-performance graphics.

### C. SIMPLE PROCESSOR USING BASIC BIT-SLICE COMPONENTS

The most basic of processors is shown in Figure 2.2. It consists of a data manipulation section, the ALU, and a control section, otherwise known as the sequencer. This basic processor will be used in this section as a framework to build a simple processor using basic bit-slice components. The Am29203 evaluation board will be used as an example of a processor using these components and will be discussed in further detail in Chapter IV. The memory section and any peripherals will be ignored for the time being.

Figure 2.3 shows a simplified view of the primary system architecture of the AM29203 evaluation board divided into the two basic sections. The ALU section of the evaluation board consists of four 4-bit 29203 data manipulation (CPU) slices to make up a 16 bit processor, and one 2904 status-and-shift control unit which is used for shift register linkage, status registers, and condition code testing. The control section of the evaluation board is made up primarily of the Am2910 and other associated hardware. The Am2910 is a 12 bit sequencer with an instruction-decoding programmed logic array provided on chip.

Looking at these basic components now in greater detail, Figure 2.4 illustrates the general structure of the manipulation unit, or the Am29203 in this specific example.

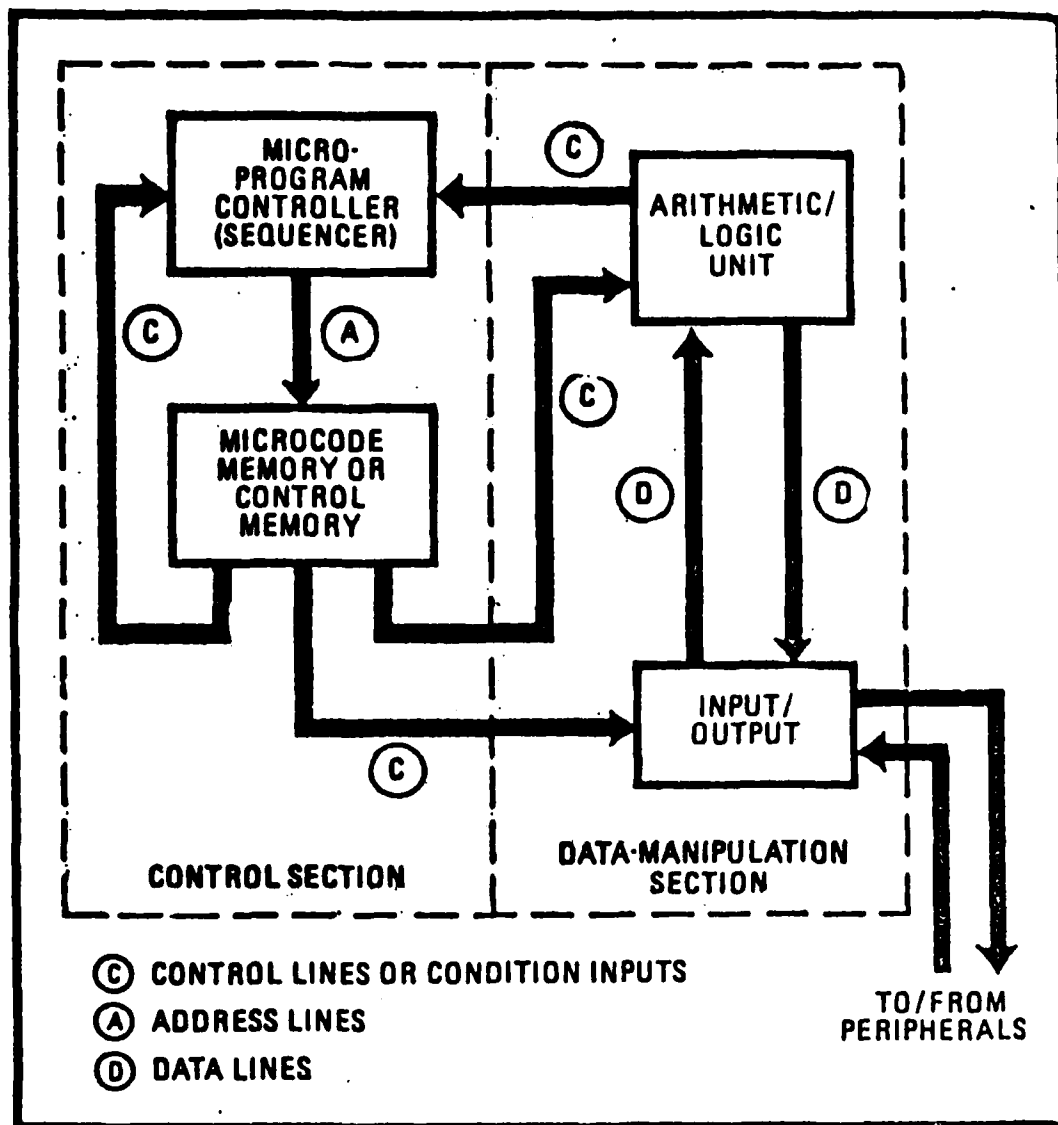


Figure 2.2 Basic Processor [Ref. 4]

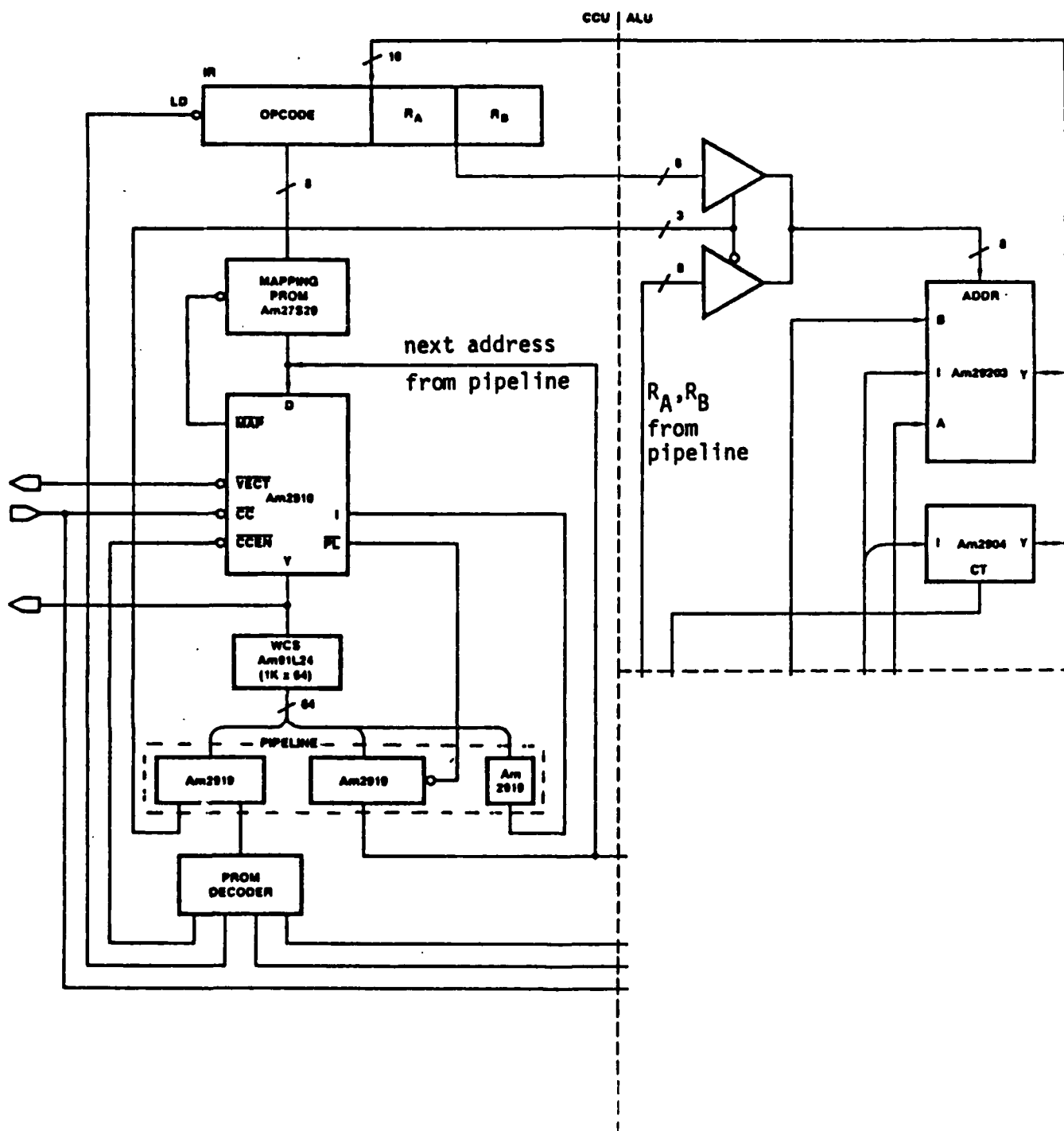


Figure 2.3 Primary System Architecture [Ref. B]

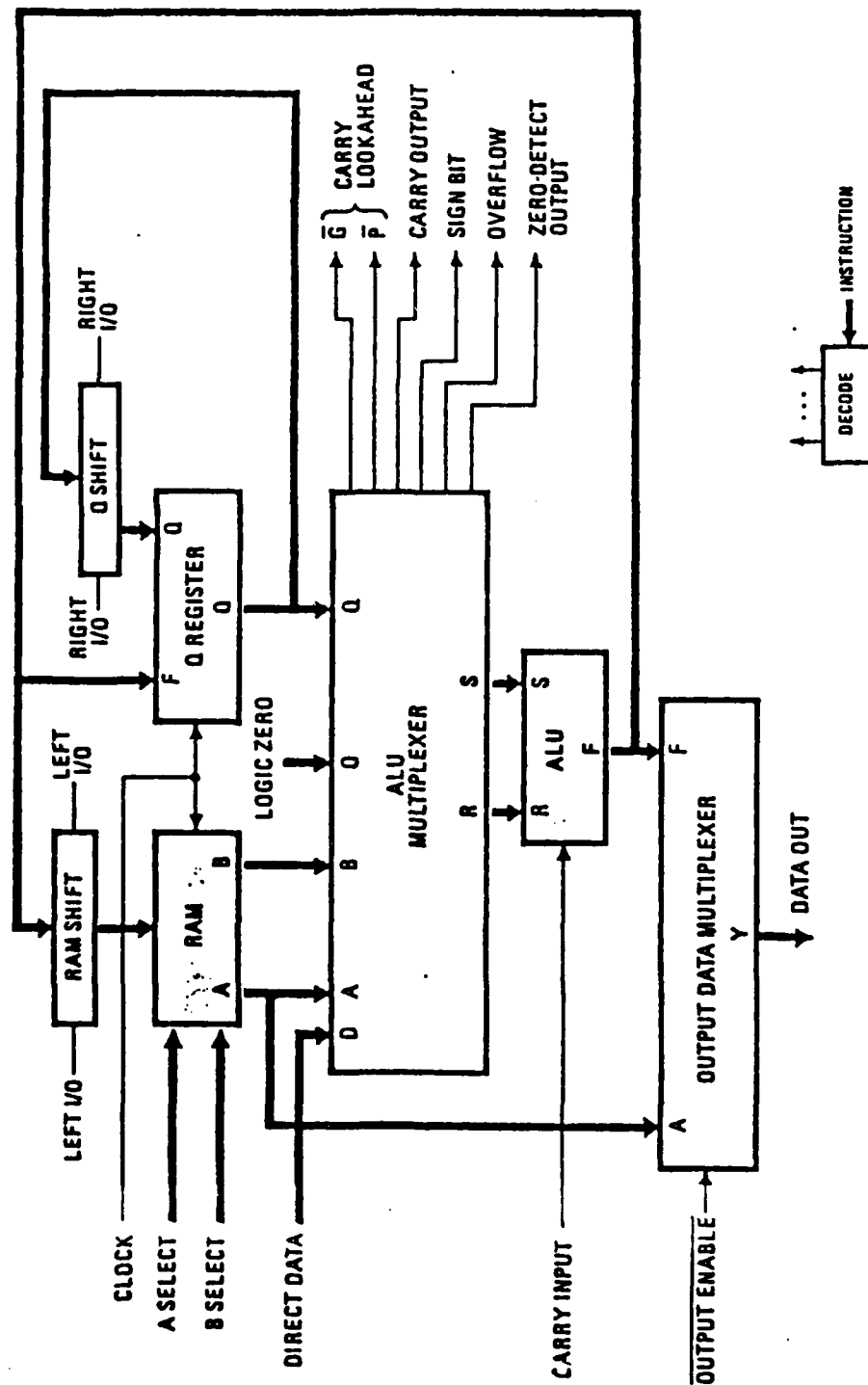


Figure 2.4 General Structure of the Am29203 [Ref. 4]



As can be seen, it consists of the ALU, for performing the required arithmetic or logic functions, general purpose registers (RAM), a multiplexer for selecting pertinent general purpose registers and a RAM shifter for performing data shifting. Of importance is the horizontal connection points shown, specifically the carry and carry look-ahead connections. Figure 2.5 illustrates how the horizontal connections are used to connect four CPU slices in a ripple carry mode to form a 16-bit ALU. This is the mode used on the evaluation board due to board space constraints and due to the fact that speed was not the primary consideration when designing the evaluation board. Had the P and G signals been connected, the processor would have been in the carry look-ahead mode, an Am2902 look-ahead carry generator would have been used, and the processor speed could therefore have been increased. This will be an important factor when looking at the time considerations later on. Also shown in Figure 2.4 are specific status conditions such as carry, sign, overflow and zero detect which are then used by the Am2904. Figure 2.6 shows the connections used between the Am29203 array and the Am2904 to allow the Am2904 to perform its status, testing and shifting functions. The Am2904 provides carry in from several sources which will also be discussed later in greater detail.

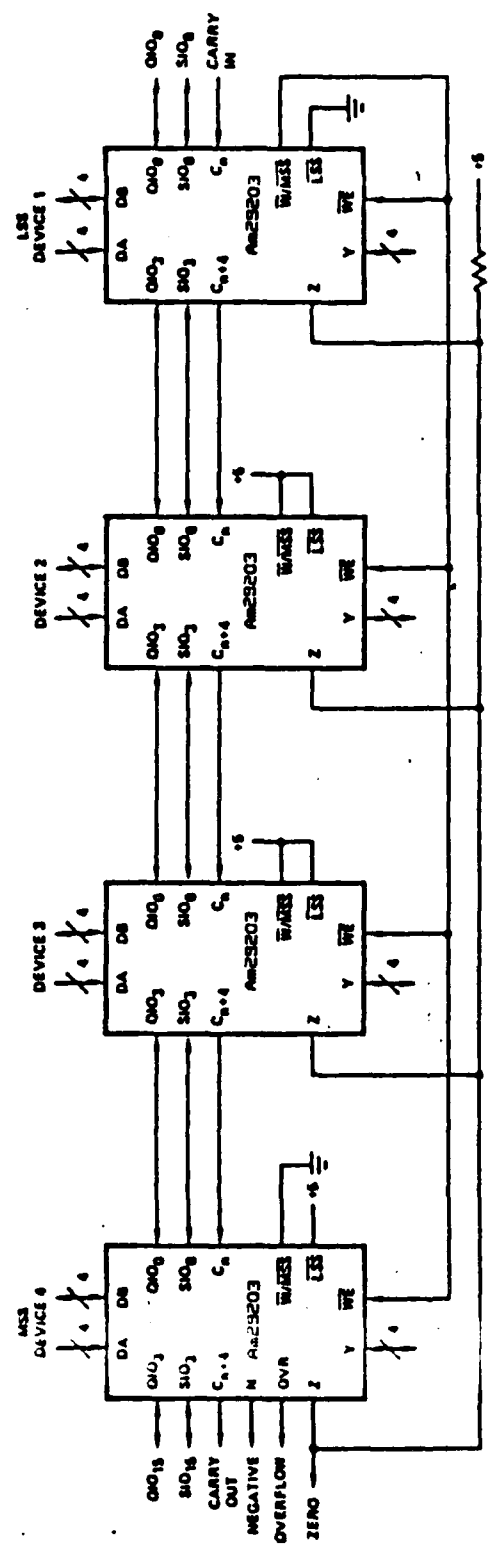


Figure 2.5 16-Bit CPU with Ripple Carry [Ref. 7:p. 7.8]

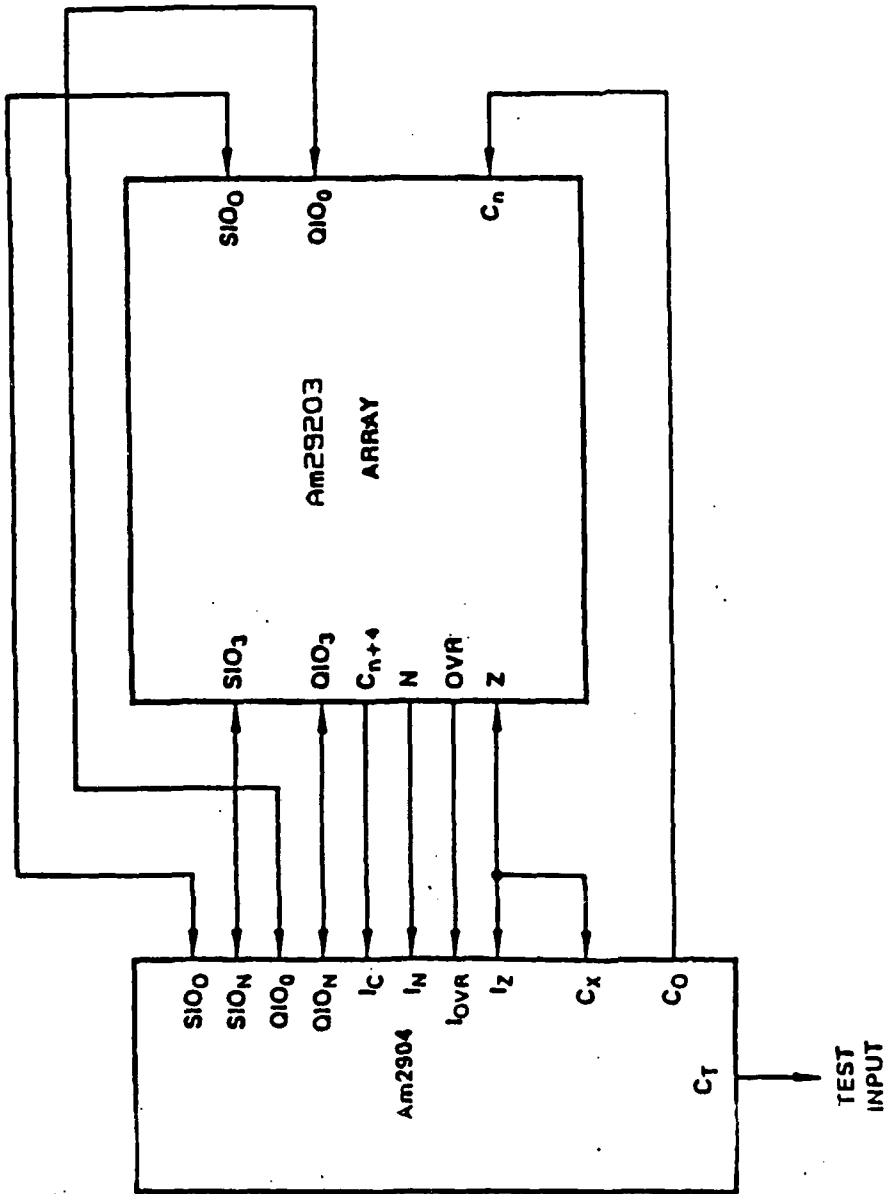


Figure 2.6 Am2904 and Am29203 Interconnections  
[Ref. 7:p. B.4]

The Am2910 as mentioned earlier, is a 12-bit sequencer used in the control section of the processor. Since it is a 12-bit sequencer, it is capable of addressing up to 4096 words of microcode, although the evaluation board only uses 10 of the 12 bits to address up to 1024 words. The function of the Am2910, put simply, is to control the sequence of execution of microinstructions. The structure of the Am2910 is as shown in Figure 2.7. From this figure it can be seen that the next address can come from four possible sources: the microprogram counter (upc), the LIFO stack (F), the register/counter (R), or from direct input through a mapping PROM. The onboard instruction PLA provides the internal controls which correspond to the next-address control logic. [Ref. 5]

Putting these Am2900 basic components together, the architecture of a 16-bit processor is as shown in Figure 2.8. It should be noted in this figure that the processor is connected in the carry look-ahead mode by interconnecting the G and P connection points. The addition of the pipeline register should also be noted. This register permits the next microinstruction to be in the process of being fetched while the current microinstruction is still executing, thereby improving the speed of the microinstruction sequencing.

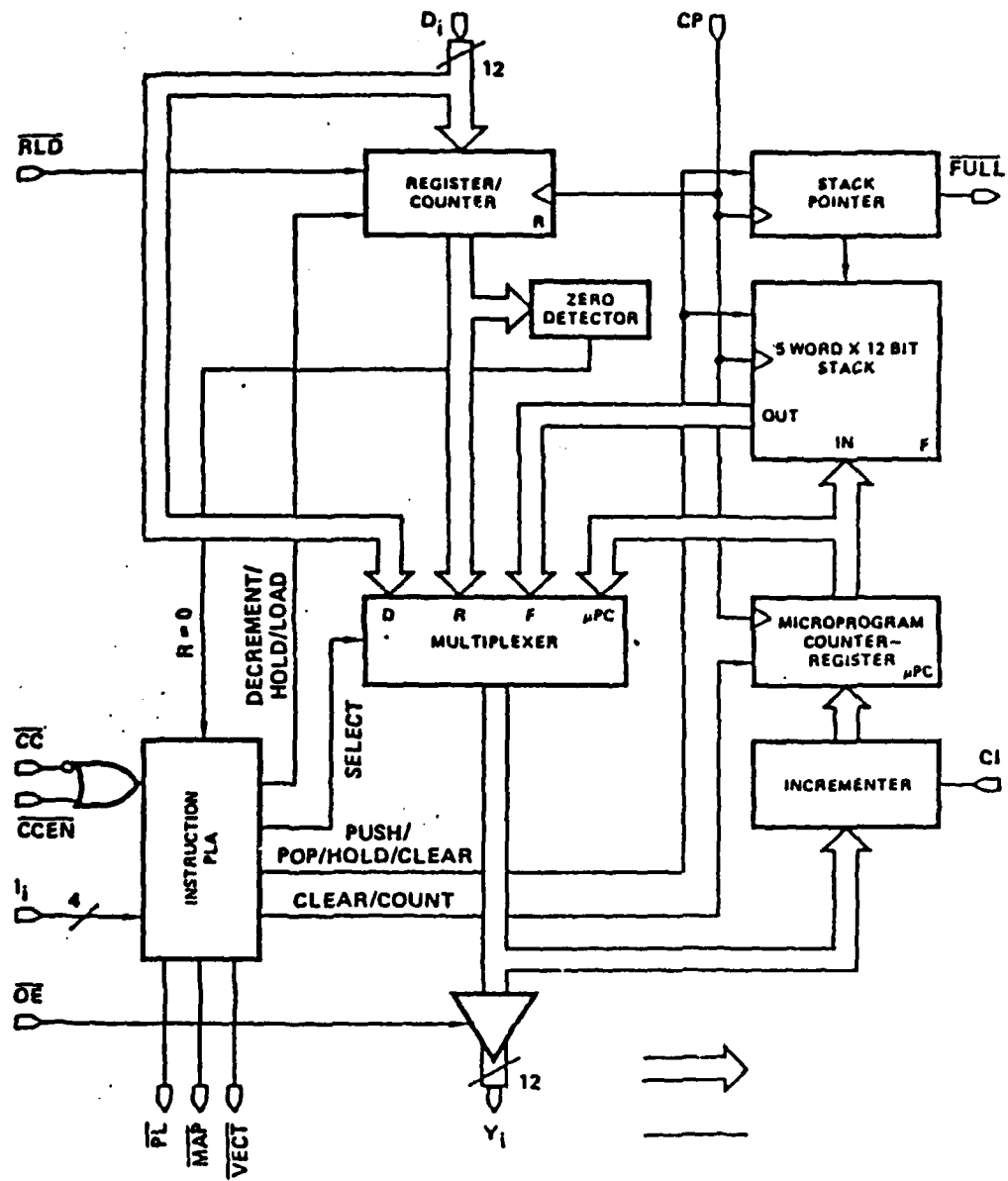


Figure 2.7 Am2910 Architecture [Ref. 7:p. 2.12]

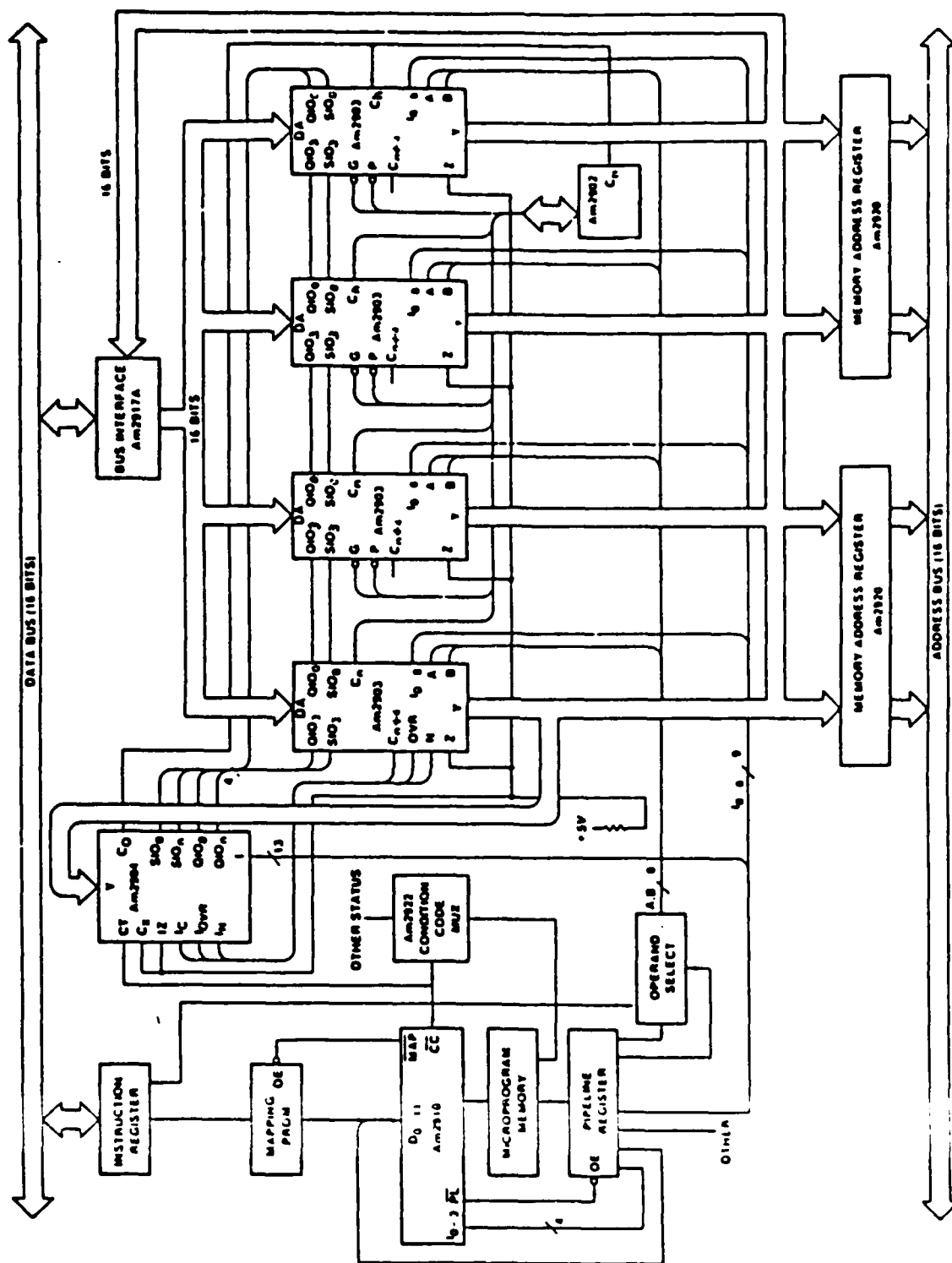


Figure 2.8 16-Bit Processor Architecture Using the Am2900 Family [Ref. 7:p 2.18]

#### D. TYPICAL MACRO AND MICRO INSTRUCTIONS

As stated earlier, the machine level or macro instructions would normally be generated by a basic compiler or assembler program. A typical format for a macro instruction is as shown in Figure 2.9. In the evaluation board, the address mode is contained in the opcode, followed by the source and destination. Suppose as an example, shown in Figure 2.10, a macro instruction mnemonic of ADDR (e.g., ADDR R1 R2) [Ref. 7:p. 2.6] is given, with the opcode given as A0 and the total macroinstruction being A012. This opcode is then mapped through a mapping PROM to give the micro-address, in this case micro-address 304, to the Am2910 microprogram sequencer.

The format of a microinstruction can vary in length from 32 to 256 bits in length (or more) depending on the amount of hardware being controlled by the microinstruction and by the presence or absence of overlaid fields. Microprogram memory (word control store-WCS) is therefore made up of relatively long words and most macroinstruction sets can be implemented in microcode using a small microprogram memory [Ref. 7:p. 2.6]. In the evaluation board, the instruction set and monitor using the instruction set are easily implemented with the 1024 WCS locations addressed by the Am2910. A typical format for a microinstruction is given by the 48-bit general microinstruction format for the

OPCODE*	ADDRESSING	SOURCE	DESTINATION
---------	------------	--------	-------------

\* address mode contained in opcode

Figure 2.9 Macroinstruction Format for Evaluation Board  
[Ref. 7:p. 3.5]

Sample Macroinstruction: A012

Mnemonic-ADDRR

A0-opcode to map to 304

1-storage address in R1

2-storage address in R2

Figure 2.10 Sample Macroinstruction



evaluation board as shown in Figure 2.11. The microinstruction is broken down into fields that control the various components. For the evaluation board, the components controlled are the Am29203, Am2904 and the Am2910 which were discussed previously. The microinstruction has several overlaid fields and even achieves what is referred to as vertical programming through the use of an overlaid command field and decoding PROM [Ref. 7:p. 3.10]. These overlaid fields make microprogramming somewhat more difficult but are used on less critical or seldom used instructions to keep the microinstruction length shorter and thereby decrease the cost of the memory (RAM) used. If speed and not cost is the primary consideration, some of these overlaid fields may have to be deleted which would then increase the word length. The coding for each of the microinstruction fields is explained in detail in the evaluation board users guide [Ref. 7]. A summary of these codes which are generally given in hexadecimal or octal form for ease of coding are shown in Figure 2.12. From this sheet for a simple 48-bit implementation, it is easily seen why a long learning process is required for complex design work using bit-slice components.

A specific example of a microinstruction is shown in Figures 2.13 and 2.14. In this example, the operation to be performed is  $R5=2*(R3+R4)$ . The codes for each field are taken from the microinstruction coding format sheet, Figure

OPERAND REGISTER ADDRESSES	ALU OPERA- TIONS	CONDITION CODES	SHIFT & CARRY	MICRO- INSTRUCTION BRANCH	NEXT ADDRESS SELECT
Am29203	Am29203	Am2904	Am2904	Am2910	Am2910

Figure 2.11 General Microinstruction Format For Evaluation Board [Ref. 7:p. 3.5]



## EVALUATION BOARD -- COOLING SHEET

Am29203										Am2904										Am2910																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
A <sub>1</sub>		A <sub>2</sub>		A <sub>3</sub>		A <sub>4</sub>		A <sub>5</sub>		A <sub>6</sub>		A <sub>7</sub>		A <sub>8</sub>		A <sub>9</sub>		A <sub>10</sub>		A <sub>11</sub>		A <sub>12</sub>		A <sub>13</sub>		A <sub>14</sub>		A <sub>15</sub>		A <sub>16</sub>		A <sub>17</sub>		A <sub>18</sub>		A <sub>19</sub>		A <sub>20</sub>		A <sub>21</sub>		A <sub>22</sub>		A <sub>23</sub>		A <sub>24</sub>		A <sub>25</sub>		A <sub>26</sub>		A <sub>27</sub>		A <sub>28</sub>		A <sub>29</sub>		A <sub>30</sub>		A <sub>31</sub>		A <sub>32</sub>		A <sub>33</sub>		A <sub>34</sub>		A <sub>35</sub>		A <sub>36</sub>		A <sub>37</sub>		A <sub>38</sub>		A <sub>39</sub>		A <sub>40</sub>		A <sub>41</sub>		A <sub>42</sub>		A <sub>43</sub>		A <sub>44</sub>		A <sub>45</sub>		A <sub>46</sub>		A <sub>47</sub>		A <sub>48</sub>		A <sub>49</sub>		A <sub>50</sub>		A <sub>51</sub>		A <sub>52</sub>		A <sub>53</sub>		A <sub>54</sub>		A <sub>55</sub>		A <sub>56</sub>		A <sub>57</sub>		A <sub>58</sub>		A <sub>59</sub>		A <sub>60</sub>		A <sub>61</sub>		A <sub>62</sub>		A <sub>63</sub>		A <sub>64</sub>		A <sub>65</sub>		A <sub>66</sub>		A <sub>67</sub>		A <sub>68</sub>		A <sub>69</sub>		A <sub>70</sub>		A <sub>71</sub>		A <sub>72</sub>		A <sub>73</sub>		A <sub>74</sub>		A <sub>75</sub>		A <sub>76</sub>		A <sub>77</sub>		A <sub>78</sub>		A <sub>79</sub>		A <sub>80</sub>		A <sub>81</sub>		A <sub>82</sub>		A <sub>83</sub>		A <sub>84</sub>		A <sub>85</sub>		A <sub>86</sub>		A <sub>87</sub>		A <sub>88</sub>		A <sub>89</sub>		A <sub>90</sub>		A <sub>91</sub>		A <sub>92</sub>		A <sub>93</sub>		A <sub>94</sub>		A <sub>95</sub>		A <sub>96</sub>		A <sub>97</sub>		A <sub>98</sub>		A <sub>99</sub>		A <sub>100</sub>		A <sub>101</sub>		A <sub>102</sub>		A <sub>103</sub>		A <sub>104</sub>		A <sub>105</sub>		A <sub>106</sub>		A <sub>107</sub>		A <sub>108</sub>		A <sub>109</sub>		A <sub>110</sub>		A <sub>111</sub>		A <sub>112</sub>		A <sub>113</sub>		A <sub>114</sub>		A <sub>115</sub>		A <sub>116</sub>		A <sub>117</sub>		A <sub>118</sub>		A <sub>119</sub>		A <sub>120</sub>		A <sub>121</sub>		A <sub>122</sub>		A <sub>123</sub>		A <sub>124</sub>		A <sub>125</sub>		A <sub>126</sub>		A <sub>127</sub>		A <sub>128</sub>		A <sub>129</sub>		A <sub>130</sub>		A <sub>131</sub>		A <sub>132</sub>		A <sub>133</sub>		A <sub>134</sub>		A <sub>135</sub>		A <sub>136</sub>		A <sub>137</sub>		A <sub>138</sub>		A <sub>139</sub>		A <sub>140</sub>		A <sub>141</sub>		A <sub>142</sub>		A <sub>143</sub>		A <sub>144</sub>		A <sub>145</sub>		A <sub>146</sub>		A <sub>147</sub>		A <sub>148</sub>		A <sub>149</sub>		A <sub>150</sub>		A <sub>151</sub>		A <sub>152</sub>		A <sub>153</sub>		A <sub>154</sub>		A <sub>155</sub>		A <sub>156</sub>		A <sub>157</sub>		A <sub>158</sub>		A <sub>159</sub>		A <sub>160</sub>		A <sub>161</sub>		A <sub>162</sub>		A <sub>163</sub>		A <sub>164</sub>		A <sub>165</sub>		A <sub>166</sub>		A <sub>167</sub>		A <sub>168</sub>		A <sub>169</sub>		A <sub>170</sub>		A <sub>171</sub>		A <sub>172</sub>		A <sub>173</sub>		A <sub>174</sub>		A <sub>175</sub>		A <sub>176</sub>		A <sub>177</sub>		A <sub>178</sub>		A <sub>179</sub>		A <sub>180</sub>		A <sub>181</sub>		A <sub>182</sub>		A <sub>183</sub>		A <sub>184</sub>		A <sub>185</sub>		A <sub>186</sub>		A <sub>187</sub>		A <sub>188</sub>		A <sub>189</sub>		A <sub>190</sub>		A <sub>191</sub>		A <sub>192</sub>		A <sub>193</sub>		A <sub>194</sub>		A <sub>195</sub>		A <sub>196</sub>		A <sub>197</sub>		A <sub>198</sub>		A <sub>199</sub>		A <sub>200</sub>		A <sub>201</sub>		A <sub>202</sub>		A <sub>203</sub>		A <sub>204</sub>		A <sub>205</sub>		A <sub>206</sub>		A <sub>207</sub>		A <sub>208</sub>		A <sub>209</sub>		A <sub>210</sub>		A <sub>211</sub>		A <sub>212</sub>		A <sub>213</sub>		A <sub>214</sub>		A <sub>215</sub>		A <sub>216</sub>		A <sub>217</sub>		A <sub>218</sub>		A <sub>219</sub>		A <sub>220</sub>		A <sub>221</sub>		A <sub>222</sub>		A <sub>223</sub>		A <sub>224</sub>		A <sub>225</sub>		A <sub>226</sub>		A <sub>227</sub>		A <sub>228</sub>		A <sub>229</sub>		A <sub>230</sub>		A <sub>231</sub>		A <sub>232</sub>		A <sub>233</sub>		A <sub>234</sub>		A <sub>235</sub>		A <sub>236</sub>		A <sub>237</sub>		A <sub>238</sub>		A <sub>239</sub>		A <sub>240</sub>		A <sub>241</sub>		A <sub>242</sub>		A <sub>243</sub>		A <sub>244</sub>		A <sub>245</sub>		A <sub>246</sub>		A <sub>247</sub>		A <sub>248</sub>		A <sub>249</sub>		A <sub>250</sub>		A <sub>251</sub>		A <sub>252</sub>		A <sub>253</sub>		A <sub>254</sub>		A <sub>255</sub>		A <sub>256</sub>		A <sub>257</sub>		A <sub>258</sub>		A <sub>259</sub>		A <sub>260</sub>		A <sub>261</sub>		A <sub>262</sub>		A <sub>263</sub>		A <sub>264</sub>		A <sub>265</sub>		A <sub>266</sub>		A <sub>267</sub>		A <sub>268</sub>		A <sub>269</sub>		A <sub>270</sub>		A <sub>271</sub>		A <sub>272</sub>		A <sub>273</sub>		A <sub>274</sub>		A <sub>275</sub>		A <sub>276</sub>		A <sub>277</sub>		A <sub>278</sub>		A <sub>279</sub>		A <sub>280</sub>		A <sub>281</sub>		A <sub>282</sub>		A <sub>283</sub>		A <sub>284</sub>		A <sub>285</sub>		A <sub>286</sub>		A <sub>287</sub>		A <sub>288</sub>		A <sub>289</sub>		A <sub>290</sub>		A <sub>291</sub>		A <sub>292</sub>		A <sub>293</sub>		A <sub>294</sub>		A <sub>295</sub>		A <sub>296</sub>		A <sub>297</sub>		A <sub>298</sub>		A <sub>299</sub>		A <sub>300</sub>		A <sub>301</sub>		A <sub>302</sub>		A <sub>303</sub>		A <sub>304</sub>		A <sub>305</sub>		A <sub>306</sub>		A <sub>307</sub>		A <sub>308</sub>		A <sub>309</sub>		A <sub>310</sub>		A <sub>311</sub>		A <sub>312</sub>		A <sub>313</sub>		A <sub>314</sub>		A <sub>315</sub>		A <sub>316</sub>		A <sub>317</sub>		A <sub>318</sub>		A <sub>319</sub>		A <sub>320</sub>		A <sub>321</sub>		A <sub>322</sub>		A <sub>323</sub>		A <sub>324</sub>		A <sub>325</sub>		A <sub>326</sub>		A <sub>327</sub>		A <sub>328</sub>		A <sub>329</sub>		A <sub>330</sub>		A <sub>331</sub>		A <sub>332</sub>		A <sub>333</sub>		A <sub>334</sub>		A <sub>335</sub>		A <sub>336</sub>		A <sub>337</sub>		A <sub>338</sub>		A <sub>339</sub>		A <sub>340</sub>		A <sub>341</sub>		A <sub>342</sub>		A <sub>343</sub>		A <sub>344</sub>		A <sub>345</sub>		A <sub>346</sub>		A <sub>347</sub>		A <sub>348</sub>		A <sub>349</sub>		A <sub>350</sub>		A <sub>351</sub>		A <sub>352</sub>		A <sub>353</sub>		A <sub>354</sub>		A <sub>355</sub>		A <sub>356</sub>		A <sub>357</sub>		A <sub>358</sub>		A <sub>359</sub>		A <sub>360</sub>		A <sub>361</sub>		A <sub>362</sub>		A <sub>363</sub>		A <sub>364</sub>		A <sub>365</sub>		A <sub>366</sub>		A <sub>367</sub>		A <sub>368</sub>		A <sub>369</sub>		A <sub>370</sub>		A <sub>371</sub>		A <sub>372</sub>		A <sub>373</sub>		A <sub>374</sub>		A <sub>375</sub>		A <sub>376</sub>		A <sub>377</sub>		A <sub>378</sub>		A <sub>379</sub>		A <sub>380</sub>		A <sub>381</sub>		A <sub>382</sub>		A <sub>383</sub>		A <sub>384</sub>		A <sub>385</sub>		A <sub>386</sub>		A <sub>387</sub>		A <sub>388</sub>		A <sub>389</sub>		A <sub>390</sub>		A <sub>391</sub>		A <sub>392</sub>		A <sub>393</sub>		A <sub>394</sub>		A <sub>395</sub>		A <sub>396</sub>		A <sub>397</sub>		A <sub>398</sub>		A <sub>399</sub>		A <sub>400</sub>		A <sub>401</sub>		A <sub>402</sub>		A <sub>403</sub>		A <sub>404</sub>		A <sub>405</sub>		A <sub>406</sub>		A <sub>407</sub>		A <sub>408</sub>		A <sub>409</sub>		A <sub>410</sub>		A <sub>411</sub>		A <sub>412</sub>		A <sub>413</sub>		A <sub>414</sub>		A <sub>415</sub>		A <sub>416</sub>		A <sub>417</sub>		A <sub>418</sub>		A <sub>419</sub>		A <sub>420</sub>		A <sub>421</sub>		A <sub>422</sub>		A <sub>423</sub>		A <sub>424</sub>		A <sub>425</sub>		A <sub>426</sub>		A <sub>427</sub>		A <sub>428</sub>		A <sub>429</sub>		A <sub>430</sub>		A <sub>431</sub>		A <sub>432</sub>		A <sub>433</sub>		A <sub>434</sub>		A <sub>435</sub>		A <sub>436</sub>		A <sub>437</sub>		A <sub>438</sub>		A <sub>439</sub>		A <sub>440</sub>		A <sub>441</sub>		A <sub>442</sub>		A <sub>443</sub>		A <sub>444</sub>		A <sub>445</sub>		A <sub>446</sub>		A <sub>447</sub>		A <sub>448</sub>		A <sub>449</sub>		A <sub>450</sub>		A <sub>451</sub>		A <sub>452</sub>		A <sub>453</sub>		A <sub>454</sub>		A <sub>455</sub>		A <sub>456</sub>		A <sub>457</sub>		A <sub>458</sub>		A <sub>459</sub>		A <sub>460</sub>		A <sub>461</sub>		A <sub>462</sub>		A <sub>463</sub>		A <sub>464</sub>		A <sub>465</sub>		A <sub>466</sub>		A <sub>467</sub>		A <sub>468</sub>		A <sub>469</sub>		A <sub>470</sub>		A <sub>471</sub>		A <sub>472</sub>		A <sub>473</sub>		A <sub>474</sub>		A <sub>475</sub>		A <sub>476</sub>		A <sub>477</sub>		A <sub>478</sub>		A <sub>479</sub>		A <sub>480</sub>		A <sub>481</sub>		A <sub>482</sub>		A <sub>483</sub>		A <sub>484</sub>		A <sub>485</sub>		A <sub>486</sub>		A <sub>487</sub>		A <sub>488</sub>		A <sub>489</sub>		A <sub>490</sub>		A <sub>491</sub>		A <sub>492</sub>		A <sub>493</sub>		A <sub>494</sub>		A <sub>495</sub>		A <sub>496</sub>		A <sub>497</sub>		A <sub>498</sub>		A <sub>499</sub>		A <sub>500</sub>		A <sub>501</sub>		A <sub>502</sub>		A <sub>503</sub>		A <sub>504</sub>		A <sub>505</sub>		A <sub>506</sub>		A <sub>507</sub>		A <sub>508</sub>		A <sub>509</sub>		A <sub>510</sub>		A <sub>511</sub>		A <sub>512</sub>		A <sub>513</sub>		A <sub>514</sub>		A <sub>515</sub>		A <sub>516</sub>		A <sub>517</sub>		A <sub>518</sub>		A <sub>519</sub>		A <sub>520</sub>		A <sub>521</sub>		A <sub>522</sub>		A <sub>523</sub>		A <sub>524</sub>		A <sub>525</sub>		A <sub>526</sub>		A <sub>527</sub>		A <sub>528</sub>		A <sub>529</sub>		A <sub>530</sub>		A <sub>531</sub>		A <sub>532</sub>		A <sub>533</sub>		A <sub>534</sub>		A <sub>535</sub>		A <sub>536</sub>		A <sub>537</sub>		A <sub>538</sub>		A <sub>539</sub>		A <sub>540</sub>		A <sub>541</sub>		A <sub>542</sub>		A <sub>543</sub>		A <sub>544</sub>		A <sub>545</sub>		A <sub>546</sub>		A <sub>547</sub>		A <sub>548</sub>		A <sub>549</sub>		A <sub>550</sub>		A <sub>551</sub>		A <sub>552</sub>		A <sub>553</sub>		A <sub>554</sub>		A <sub>555</sub>		A <sub>556</sub>		A <sub>557</sub>		A <sub>558</sub>		A <sub>559</sub>		A <sub>560</sub>		A <sub>561</sub>		A <sub>562</sub>		A <sub>563</sub>		A <sub>564</sub>		A <sub>565</sub>		A <sub>566</sub>		A <sub>567</sub>		A <sub>568</sub>		A <sub>569</sub>		A <sub>570</sub>		A <sub>571</sub>		A <sub>572</sub>		A <sub>573</sub>		A <sub>574</sub>		A <sub>575</sub>		A <sub>576</sub>		A <sub>577</sub>		A <sub>578</sub>		A <sub>579</sub>		A <sub>580</sub>		A <sub>581</sub>		A <sub>582</sub>		A <sub>583</sub>		A <sub>584</sub>		A <sub>585</sub>		A <sub>586</sub>		A <sub>587</sub>		A <sub>588</sub>		A <sub>589</sub>		A <sub>590</sub>		A <sub>591</sub>		A <sub>592</sub>		A <sub>593</sub>		A <sub>594</sub>		A <sub>595</sub>		A <sub>596</sub>		A <sub>597</sub>		A <sub>598</sub>		A <sub>599</sub>		A <sub>600</sub>		A <sub>601</sub>		A <sub>602</sub>		A <sub>603</sub>		A <sub>604</sub>		A <sub>605</sub>		A <sub>606</sub>		A <sub>607</sub>		A <sub>608</sub>		A <sub>609</sub>		A <sub>610</sub>		A <sub>611</sub>		A <sub>612</sub>		A <sub>613</sub>		A <sub>614</sub>		A <sub>615</sub>		A <sub>616</sub>		A <sub>617</sub>		A <sub>618</sub>		A <sub>619</sub>		A <sub>620</sub>		A <sub>621</sub>		A <sub>622</sub>		A <sub>623</sub>		A <sub>624</sub>		A <sub>625</sub>		A <sub>626</sub>		A <sub>627</sub>		A <sub>628</sub>		A <sub>629</sub>		A <sub>630</sub>		A <sub>631</sub>		A <sub>632</sub>		A <sub>633</sub>		A <sub>634</sub>		A <sub>635</sub>		A <sub>636</sub>		A <sub>637</sub>		A <sub>638</sub>		A <sub>639</sub>		A <sub>640</sub>		A <sub>641</sub>		A <sub>642</sub>		A <sub>643</sub>		A <sub>644</sub>		A <sub>645</sub>		A <sub>646</sub>		A <sub>647</sub>		A <sub>648</sub>		A <sub>649</sub>		A <sub>650</sub>		A <sub>651</sub>		A <sub>652</sub>		A <sub>653</sub>		A <sub>654</sub>		A <sub>655</sub>		A <sub>656</sub>		A <sub>657</sub>		A <sub>658</sub>		A <sub>659</sub>		A <sub>660</sub>		A <sub>661</sub>		A <sub>662</sub>		A <sub>663</sub>		A <sub>664</sub>		A <sub>665</sub>		A <sub>666</sub>		A <sub>667</sub>		A <sub>668</sub>		A <sub>669</sub>		A <sub>670</sub>		A <sub>671</sub>		A <sub>672</sub>		A <sub>673</sub>		A <sub>674</sub>		A <sub>675</sub>		A <sub>676</sub>		A <sub>677</sub>		A <sub>678</sub>		A <sub>679</sub>		A <sub>680</sub>		A <sub>681</sub>		A <sub>682</sub>		A <sub>683</sub>		A <sub>684</sub>		A <sub>685</sub>		A <sub>686</sub>		A <sub>687</sub>		A <sub>688</sub>		A <sub>689</sub>		A <sub>690</sub>		A <sub>691</sub>		A <sub>692</sub>		A <sub>693</sub>		A <sub>694</sub>		A <sub>695</sub>		A <sub>696</sub>		A <sub>697</sub>		A <sub>698</sub>		A <sub>699</sub>		A <sub>700</sub>		A <sub>701</sub>		A <sub>702</sub>		A <sub>703</sub>		A <sub>704</sub>		A <sub>705</sub>		A <sub>706</sub>		A <sub>707</sub>		A <sub>708</sub>		A <sub>709</sub>		A <sub>710</sub>		A <sub>711</sub>		A <sub>712</sub>		A <sub>713</sub>		A <sub>714</sub>		A <sub>715</sub>		A <sub>716</sub>		A <sub>717</sub>		A <sub>718</sub>		A <sub>719</sub>		A <sub>720</sub>		A <sub>721</sub>		A <sub>722</sub>		A <sub>723</sub>		A <sub>724</sub>		A <sub>725</sub>		A <sub>726</sub>		A <sub>727</sub>		A <sub>728</sub>		A <sub>729</sub>		A <sub>730</sub>		A <sub>731</sub>		A <sub>732</sub>	

### Figure 2.13 Sample Instruction Entries to Coding Sheet

BITS	VALUE	EXPLANATION
47-45	Q#4	Sources Ra & Rb specified by pipeline, destination Rc specified by IR
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#0	Operand Sources from RAM
39-36	H#8	Destination to RAM with arithmetic upshift
35-32	H#3	ADD, Rc=Ra + Rb
31-30	B#00	No carry in
29-24	Q#20	ALU status to status registers
23	B#1	Don't latch micro status
22	B#0	Latch macro status
21	B#1	No command enable
20	B#0	Shift enable
19-16	H#2	Up shift, zero fill
15	B#1	Don't set breakpoint
14	X	Spare/Don't care
13-12	B#XX	Don't care
11-8	H#3	Ra-R3
7-4	H#4	Rb-R4
3-0	H#A	Conditional Return

Resulting Microword: 8083 10A2 F34A

Perform  $R5-2*(R3+R4)$  with sources specified by pipeline and destination specified by IR

NOTE: B=Binary, Q=Octal, H=Hexadecimal

Figure 2.14 Sample Microword With Field Descriptions

2.12, and transferred to a blank coding sheet as demonstrated in Figure 2.13. These codes are formed into a 12-element hexadecimal word which is then explained in Figure 2.14. For instance, the octal code #4 is placed in bits 47-45 which translates to the sources Ra and Rb being specified by the pipeline and the destination being specified by the instruction register (IR). The pipeline field, bits 11-4, then designates Ra and Rb to be registers R3 and R4 respectively. The addition function is performed by the ALU by specifying code hexadecimal #3 in the ALU function field, bits 35-32, while the multiply by 2 is implemented using the AM2904 shifter. The codes for the shifting are placed in the Am2904 field and the micro status is latched for possible overflow. The Am2910 instruction in this case is a conditional return (based on the condition of the status registers) and is performed by placing the hexadecimal #A in the Am2910 instruction field, bits 3-0. The resulting 12 element hexadecimal microword is as shown. Typically, several of these microinstructions would be used to implement a single macro instruction.

#### E. BIT-SLICE: METHODOLOGY OR DEVICE

Some people today believe that bit-slice is an outdated device. The argument to be presented here is that a device will be outdated as technology improves whereas a method should be updated with advances in technology. Indeed, if bit-slice were associated with a device, then bit-slice

components, which were first conceived in 1974, should have long been replaced by other devices and components, considering the rapid developments in recent technology. However, as technology has increased, bit-slice devices have continued to improve and the demand for these components has continued to grow. The following paragraphs will give some specific examples of recent advances in the bit-slice method.

Probably the most widely used application of bit-slice is that of its use in high-performance graphics, due to the high speed required to process large amounts of data. An example of this is found hidden in Ramtek's graphic display system which uses the Am2910 sequencer for its memory control processor [Ref. 8]. Although VLSI technology recently brought about powerful graphic controller chips, this same technology has also improved the performance of the bit-slice. While the VLSI chips have the advantage of low cost for high volume and capabilities for a non-standard bus, the advantages of the bit-slice over the VLSI chips are:

- very high writing speeds,
- support of graphics standards, and
- programmability.

This last item may be the distinct advantage in that it:

- permits graphics interface to be tuned to the particular requirements of the application,
- can be programmed to emulate existing graphics devices,

- can easily accommodate field changes or upgrades,
- specialized graphics operations may be microcoded, moving intensive computational loads from the host processor to the bit-slice, and
- easily adapts to changing graphics standards. [Ref. 9]

Texas Instruments introduced its STL 8-bit slice microprocessor parts in 1985 and ECL 8-bit slice microprocessor parts in 1986 using IMPACT (implanted advanced composed technology). The STL devices enabled STL circuitry to match conventional ECL gate delays but at a thirtieth the power while the ECL devices cut ECL gate delay three to four times with conventional ECL power dissipation. This architecture raised throughput significantly as the processor can read an address, perform an ALU operation, and shift and write all in the period of a single clock cycle. [Ref. 10]

LSI Logic Corporation has made a recent introduction to the semi-custom market using on-board bit-slice methods in its design of structured arrays for microprogrammed systems. These structured arrays can approach the density of full-custom design circuits while retaining the quick design turnaround time of gate arrays. The LSA devices combine up to eight 2901s, 64K of ROM and 3900 gates of logic array on a single chip. In a typical application of these devices it was shown that a single chip could be used to replace 59 discrete 2900-family devices with a power consumption



reduction from 40 W to 1.5 W and a 50% increase in processor performance. [Ref. 11]

The final example given is the introduction by VITESSE Electronic Corporation of 2900 Bit-Slice components offered in Gallium Arsenide chips. These devices were the first commercial devices to be offered in Gallium Arsenide. Using enhancement-depletion mode chips to solve earlier depletion-mode Gallium Arsenide design problems, VITESSE was able to achieve low cost production of these devices using a silicon-like fabrication process. With amazing gate delays in the range of 125 picoseconds ( $1/8$  of a nanosecond), VITESSE easily achieved speeds of 13-ns for a 4-bit add and a RAM 3.5-ns cycle time using a conservative design approach. Compared to AMD's high speed ECL 2900 components, the Gallium Arsenide components can run at speeds two to three times faster. This example is probably the most convincing argument that bit-slice is not an outdated device but rather a methodology which has continued to improve with technological advances. [Ref. 1]

### III. FORTRAN IMPLEMENTATION OF FIR FILTER

#### A. INTRODUCTION OF FIR DIGITAL FILTER

The filter chosen to be implemented in bit-slice was an FIR (Finite-impulse-response) digital filter. This type of filter offers many advantages. First, since it is FIR, it can always be made to be stable and causal [Ref. 12]. Secondly, since it is digital, it possesses the inherent advantage of immunity to noise and can be subjected to error detecting codes, thereby offering a high reliability not found in analog signals. As will be shown later in this chapter, the accuracy of a digital signal can be increased by increasing the number of bits used in the data stream and software or hardware implementation. Further advantages of the digital filter are that it can be easily duplicated for precise processing, with fine tuning of analog components replaced by data and program manipulation for consistent output. With this precision, large amounts of data can be processed with error detecting comparisons possible. The digital signals used can be stored for long or short periods of time without loss of accuracy. All of these advantages come with the price of noise introduced due to quantization, which will also be discussed in this chapter. Finally, the cost and size of these highly reliable and accurate digital

filters are greatly reduced from their expensive analog counterparts. [Ref. 12]

The specific filter chosen to be implemented in bit-slice was a clever video processor filter as shown in Figure 3.1, with an advertized bandpass color subcarrier frequency of 3.58MHz and a sampling frequency four times the subcarrier frequency, or 14.32MHz. This filter is shown below in equation (z-domain) form:

$$H(Z) = (1-Z^{-2})(1+Z^{-4})(1+Z^{-3})(1-Z^{-1})(1-Z^{-1})(1-Z^{-2})$$

This filter has the distinct advantage of using only coefficients of 1 in each of its six stages which allows the filter to be designed using simple shift and add circuits. Reference 2 neither states or derives how this 13th order filter was reduced to its six stages nor does it explain why the stages were ordered in the manner in which they were ordered. Mathematically, it does not matter which order the stages are put in. However, in the real environment, it may be possible that this particular ordering of the stages offers some advantage. These issues were looked at only briefly as will be mentioned in the quantization section of this chapter, however, a possible follow-on thesis may explore these issues more fully.

#### B. "DSL" PROGRAM IMPLEMENTATION

Initially, to obtain a better understanding of this filter, the six stages were multiplied together to obtain

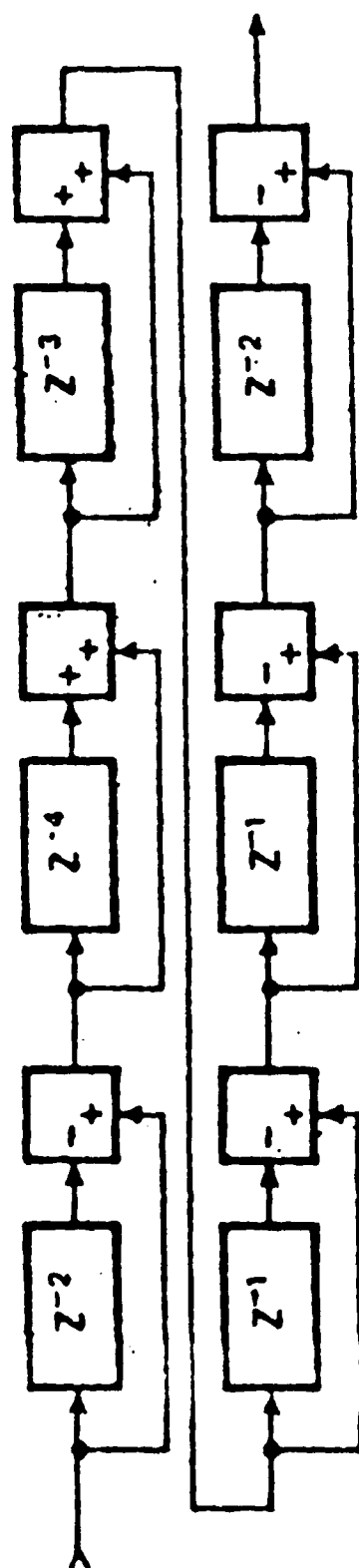


Figure 3.1 FIR Digital Filter [Ref. 2]

the rational polynomial form and factored or cascaded form as shown below:

Rational Polynomial Form:

$$H(Z) = Z^{-13} (Z^{13} - 2Z^{12} - Z^{11} + 5Z^{10} - 2Z^9 - 5Z^8 + 4Z^7 + 4Z^6 - 5Z^5 - 2Z^4 + 5Z^3 - Z^2 - 2Z + 1)$$

Factored or Cascaded Form:

$$H(Z) = Z^{-13} (Z-1)^4 (Z+1)^3 (Z+.707+j.707) (Z-.707+j.707) (Z+.707-j.707) (Z-.707-j.707) (Z-.5+j.866) (Z-.5-j.866)$$

These forms were used to obtain the required data entry to utilize a student-designed graphing program entitled "controls," on the IBM mainframe. Although this program provided the expected magnitude frequency response, it appeared to be too difficult to use to obtain desired signal input/output graphs. Another program entitled "DSL" (Dynamic Simulation Language), as provided by IBM in their language reference manual and installed on the mainframe, was then used. This program provided a more versatile plotting of the magnitude-frequency response of the filter and not only allowed the filter to be entered in its coefficient form but also in its original six-stage form as well. The "DSL" program proved to be a very useful tool in the way of a quick visual reference of signal input/output to the filter and was used continuously throughout the thesis development.

First, "DSL" was used to obtain the magnitude-frequency response as shown in Figure 3.2. The procedure and program for obtaining this graph is shown in Figure 3.3. Indeed, the frequency response for a bandpass filter is obtained as expected. With THETA from the graph equal to PI, the following is found to be true:

With  $f$ =input frequency and  $F_s$ =sampling frequency

$$f = \text{THETA} * F_s / 2$$

For Center Frequency of Passband:

$$f = .575 * 14.32 \text{ MHz} / 2 = 4.10 \text{ MHz}$$

For Subcarrier Frequency of the Passband:

$$f = .500 * 14.32 \text{ MHz} / 2 = 3.58 \text{ MHz}$$

Therefore, the center frequency of the passband is found to be 4.1 MHz and the subcarrier frequency of 3.58 MHz is slightly below the center of the passband, both as predicted by Reference 2. The "DSL" program was then used to obtain input/output graphs using the rational polynomial form of the filter as shown in Figure 3.4. In this particular implementation and throughout the rest of the implementations, a standard sine function was used for the input to the filter. Figures 3.5, 3.6 and 3.7 show output responses for inputs below, within, and above the passband respectively as indicated. Again, as expected, the output was zero (steady state) for an input below the passband. The output for the in-band subcarrier frequency of 3.58 MHz

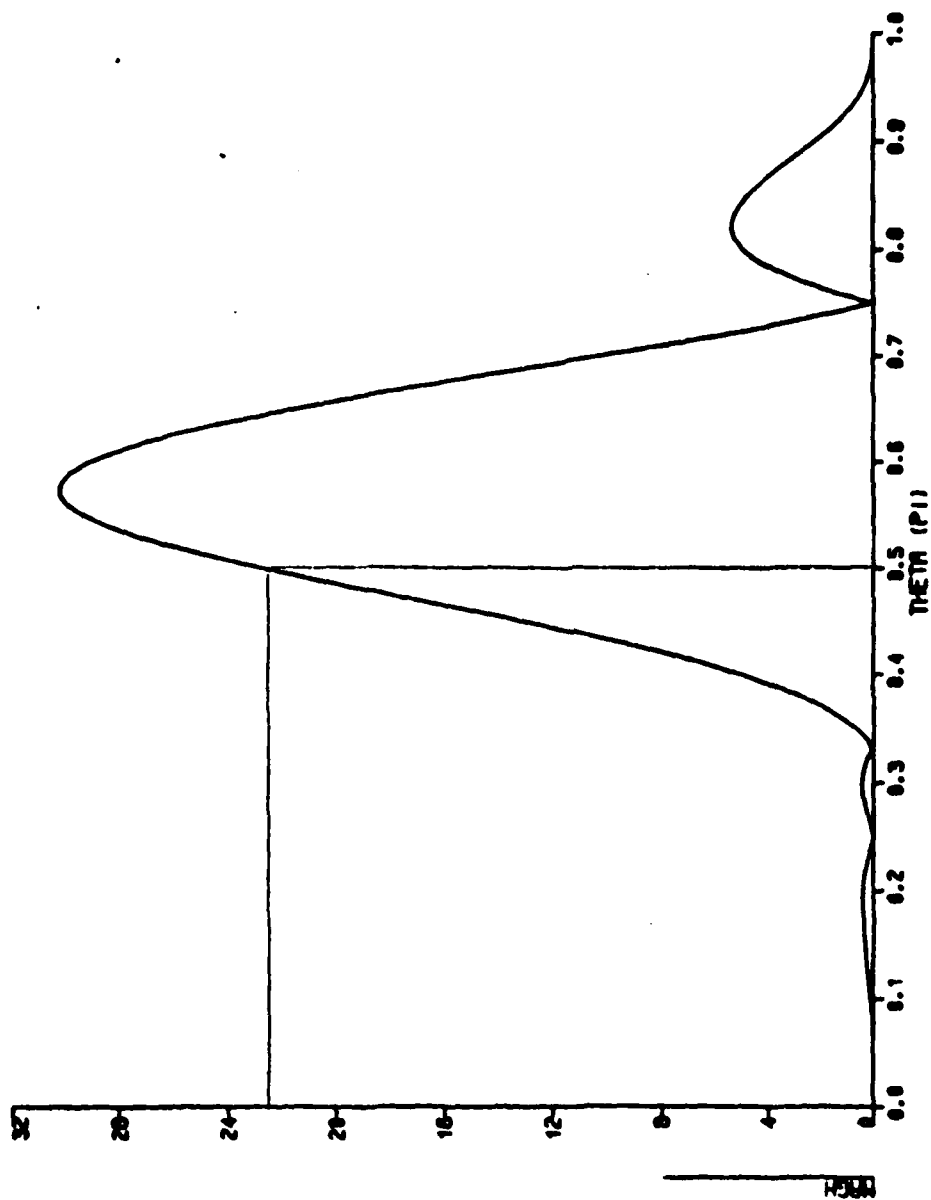


Figure 3.2 DSL Magnitude-Frequency Response of FIR Filter

```

TITLE DIGITAL FILTER
***** TO USE THIS PROGRAM, DO THE FOLLOWING STEPS:
*      1. BE AT A TEK618 GRAPHICS TERMINAL *      (YOU ONLY NEED
*      2. TYPE "CP DEFINE STORAGE 1500K"      *      TO DO THESE FOUR
*      3. TYPE "I CMS"                        *      WHEN YOU FIRST
*      4. TYPE "LINKTO DSL"                    *      LOG ON... )
**      RUNNING THE PROGRAM *****
*      5. GO INTO XEDIT AND MODIFY, IF NECESSARY, YOUR VALUES
*          OF THE FILTER COEFFICIENTS.
*      6. NOW YOU CAN RUN AS MANY TIMES AS YOU WANT. TO RUN THE
*          PROGRAM, TYPE "DSL DIGITA FORTRAN A1 (G"
*
COMPLEX S,H,H1,H2,H3,H4
CONST A1=-2.0,A2=-1.0,A3=5.0,A4=-2.0,A5=-5.0,A6=4.0,A7=4.0
CONST A8=-5.0,A9=-2.0,A10=5.0,A11=-1.0,A12=-2.0,A13=1.0
*      K=(1./(1+A1+A2+A3+A4+A5+A6+A7+A8+A9+A10+A11+A12+A13))
      THET=THETA*PI
      S=CMPLX(0.,THET)
      H1=A1*CEXP(-S)+A2*CEXP(-2*S)+A3*CEXP(-3*S)+A4*CEXP(-4*S)
      H2=A5*CEXP(-5*S)+A6*CEXP(-6*S)+A7*CEXP(-7*S)+A8*CEXP(-8*S)
      H3=A9*CEXP(-9*S)+A10*CEXP(-10*S)+A11*CEXP(-11*S)+A12*CEXP(-12*S)
      H4=A13*CEXP(-13*S)+CEXP(S*0.)
      H=H1+H2+H3+H4
      SHIFT=RADEG*PHASE(0.,H)
      MAGH=10**GAIN(H)
RENAME TIME=THETA
CONTROL FINTIM=1.00,DELT=.01
PRINT .1,MAGH,SHIFT
SAVE .01,MAGH,SHIFT
GRAPH (DE=TEK618) THETA(UN=PI RADIANS),MAGH
GRAPH (DE=TEK618) THETA(UN=PI RADIANS),SHIFT(UN=DEGREES)
LABEL FREQUENCY RESPONSE MAGNITUDE OF FIR DIGITAL FILTER
LABEL PHASE SHIFT PLOT FOR FIR DIGITAL FILTER
END
STOP

```

Figure 3.3 DSL Program Entry Instructions and Magnitude-Frequency Response Program



```

TITLE DIGITAL FILTER(REAL TIME RESPONSE)
INITIAL Y=0.
INITIAL X1=0. ,X2=0. ,X3=0. ,X4=0. ,X5=0. ,X6=0. ,X7=0. ,X8=0. ,X9=0. ,X10=0.
INITIAL X11=0. ,X12=0. ,X13=0.
INITIAL X=0.
CONST A1=-2.0,A2=-1.0,A3=5.0,A4=-2.0,A5=-5.0,A6=4.0,A7=4.0
CONST A8=-5.0,A9=-2.0,A10=5.0,A11=-1.0,A12=-2.0,A13=1.0
CONST B=1.0
CONST F=3.58E5
CONST FS=1.432E7
DYNAMIC
  X13=X12
  X12=X11
  X11=X10
  X10=X9
  X9=X8
  X8=X7
  X7=X6
  X6=X5
  X5=X4
  X4=X3
  X3=X2
  X2=X1
  X1=X
  TIME1=K/FS
  THETA=2.*PI*F*TIME1
  X=B*SIN(THETA)
  Y=X+A1*X1+A2*X2+A3*X3+A4*X4+A5*X5+A6*X6+A7*X7+A8*X8+A9*X9...
    +A10*X10+A11*X11+A12*X12+A13*X13
RENAME TIME=K
CONTROL FINTIM=100,DELT=1.
PRINT 1.,TIME1,X,Y
SAVE 1.,TIME1,X,Y
GRAPH (DE=TEK618) TIME1(UN=SECS),Y(MA=5)
GRAPH (DE=TEK618) TIME1(UN=SECS),X(MA=4)
LABEL OUTPUT OF DIGITAL FILTER
LABEL INPUT TO DIGITAL FILTER
END
STOP

```

Figure 3.4 DSL Program of FIR Filter in Rational Polynomial Form

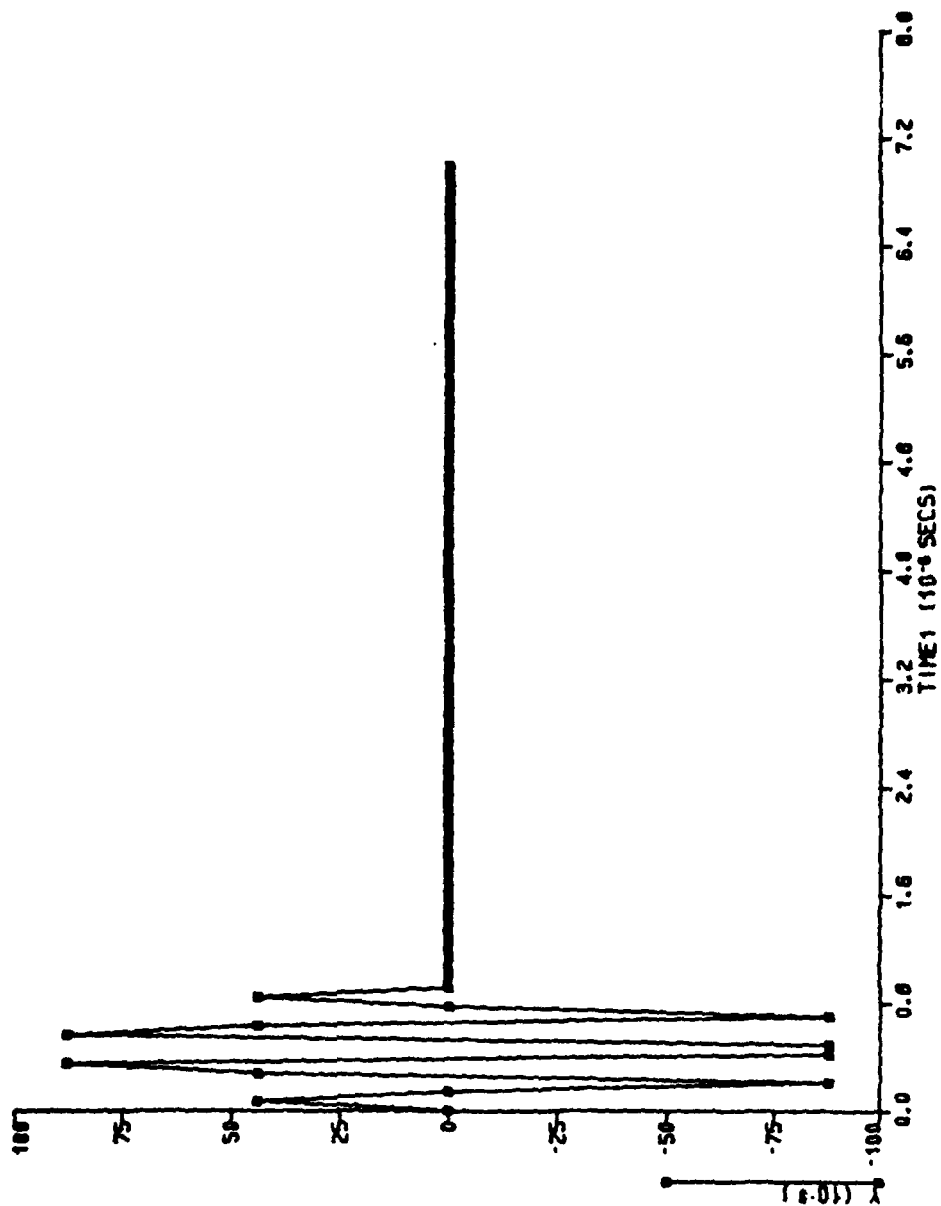


Figure 3.5 DSL Graph of Output Using Sine Wave Input at Frequency (F=105) Below FIR Filter Passband

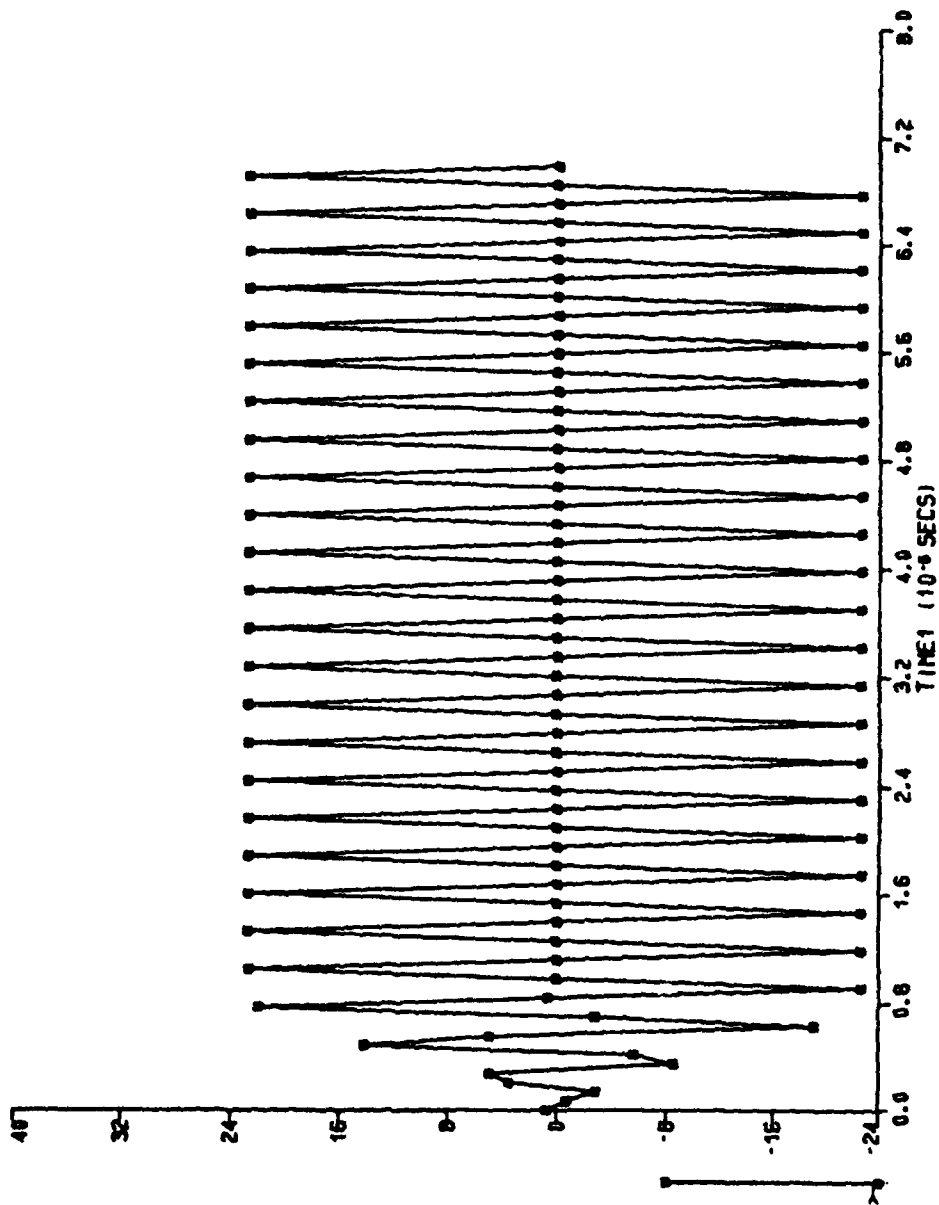


Figure 3.6 DSL Graph of Output Using Sine Wave Input at Frequency (F=3.58\*106) in FIR Filter Passband

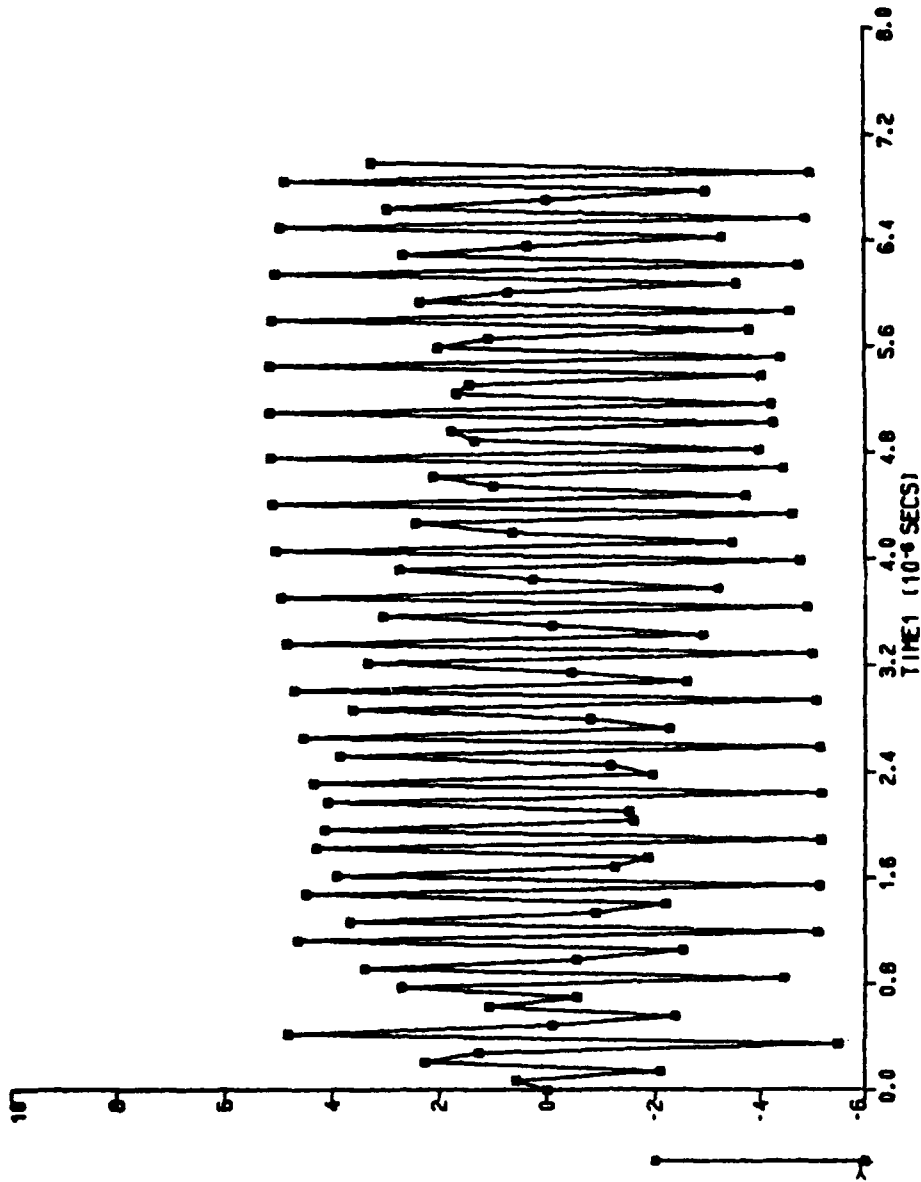


Figure 3.7 DSL Graph of Output Using Sine Wave Input at Frequency (F-1012) above FIR Filter Passband)

passed through the filter with a gain of approximately 23 as predicted by the magnitude-frequency response (Figure 3.2). And finally for an input above the passband, the output showed aliasing in the computer environment as the sampling frequency is no longer at least twice the input frequency.

### C. FORTRAN IMPLEMENTATION

The next step in preparation for implementing this filter in bit-slice was to implement the filter in Fortran on the VAX mainframe. The original concept was that once the Fortran version of the filter was working, the VAX command "Fortran/List/Machine\_Code 'File Name'" [Ref. 13] would then be used to obtain the program file in a form similar to the VAX macro assembly listing. The purpose of obtaining this assembly code was to implement the filter at the assembly language level or at least gain some insight as to how the filter might be better implemented in bit-slice. These "macro" level commands turned out to be too straight forward for the "micro" level language of the bit-slice, especially when considering the use of the registers for the shifting, as will be demonstrated in the next chapter.

The six stage shift-and-add form of the filter was used with the variables added to Figure 3.1 as shown in Figure 3.8. The equations for this implementation are as follows:

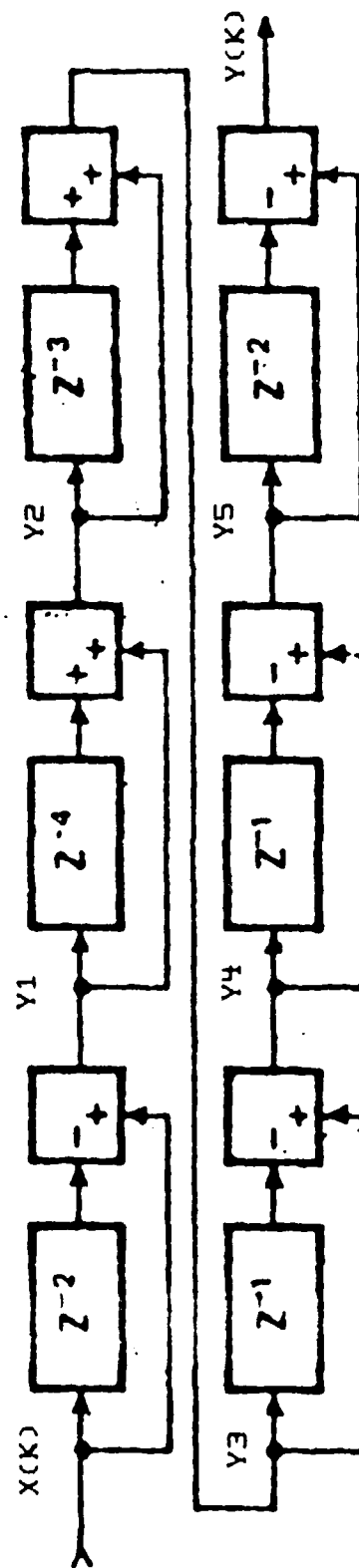


Figure 3.8 FIR Digital Filter with Variables Added

<b>Stage 1</b>	<b><math>Y1=X(K)-X2</math></b> $X2=X1$ $X1=X(K)$
<b>Stage 2</b>	<b><math>Y2=Y1+Y14</math></b> $Y14=Y13$ $Y13=Y12$ $Y12=Y11$ $Y11=Y1$
<b>Stage 3</b>	<b><math>Y3=Y2+Y23</math></b> $Y23=Y22$ $Y22=Y21$ $Y21=Y2$
<b>Stage 4</b>	<b><math>Y4=Y3-Y31</math></b> $Y31=Y3$
<b>Stage 5</b>	<b><math>Y5=Y4-Y41</math></b> $Y41=Y4$
<b>Stage 6</b>	<b><math>Y(K)=Y5-Y52</math></b> $Y52=Y51$ $Y51=Y5$

For ease of understanding the equations, each of the six adder stages are printed in bold face type. The equations which follow the adder equations are used to obtain values for the unit-time delay variables. For example, in the Stage 1 adder equation, the variable  $X2$  represents the value of  $X(K)$  delayed two units of time. To obtain the value for  $X2$ , the two equations which follow the Stage 1 adder equation are used, as shown in an example in Figure 3.9. In this example, at time  $t$ ,  $X(K)$  is equal to 5. Two units of time later, at time  $t+2$ , the value of 5 has been 'shifted' to the variable  $X2$  in the adder equation.

A structured Fortran programming approach was used to implement the filter, at this point in the development, with the program as shown in Figure 3.10. This approach offered many advantages. First, by breaking the different components of the program into a main calling routine,

Data Sequence

time t:  $X(K)=5$   
time t+1:  $X(K)=8$   
time t+2:  $X(K)=10$

Initial Conditions

$X1=0$   
 $X2=0$

Time Sequence

<u>t</u>	<u>t+1</u>	<u>t+2</u>
$Y1=X(K)-X2$ =5-0	$Y1=X(K)-X2$ =8-0	$Y1=X(K)-X2$ =10-5
$X2=X1$ =0	$X2=X1$ =5	$X2=X1$ =8
$X1=X(K)$ =5	$X1=X(K)$ =8	$X1=X(K)$ =10

Figure 3.9 Example of Stage 1 Equations and Numerical Representations for Time t Through Time t+2



```

C      THIS PROGRAM IS A REPRESENTATION OF A 13TH ORDER BAND PASS FILTER
C
      REAL *8 X(100),Y(100),T(101)
      INTEGER N
      PRINT 4
4      FORMAT ('1')
      CALL INPUT (N,X,T)
      CALL FUNCT (X,Y,N)
      CALL OUTPUT (X,Y,T,N)
      PRINT 4
      STOP
      END

C -----
      SUBROUTINE INPUT (N,X,TIME1)
      REAL *8 X(100),F,FS,TIME1(101)
      INTEGER N,K
      N=100
      F=3.58E6
      FS=1.432E7
      TIME1(1)=0.
      DO 100 K=1,N
          THETA=2.*3.1415926*F*TIME1(K)
          X(K)=SIN(THETA)
          TIME1(K+1)=K/FS
100      CONTINUE
      RETURN
      END

C -----
      SUBROUTINE FUNCT (X,Y,N)
      REAL *8 X(100),Y(100)
      REAL *8 X1/O./,X2/O./,Y14/O./,Y13/O./,Y12/O./,Y11/O./,Y23/O./
      REAL *8 Y22/O./,Y21/O./,Y31/O./,Y41/O./,Y52/O./,Y51/O./
      INTEGER K,N
      DO 50 K=1,N
          Y1=X(K)-X2
          X2=X1
          X1=X(K)
          Y2=Y1+Y14
          Y14=Y13
          Y13=Y12
          Y12=Y11
          Y11=Y1
          Y3=Y2+Y23
          Y23=Y22
          Y22=Y21
          Y21=Y2
          Y4=Y3-Y31
          Y31=Y3
          Y5=Y4-Y41
          Y41=Y4
          Y(K)=Y5-Y52
          Y52=Y51
          Y51=Y5
50      CONTINUE
      RETURN
      END

C -----
      SUBROUTINE OUTPUT (X,Y,T,N)
      REAL *8 X(100),Y(100),T(101)
      INTEGER I,N
      DO 200 I=1,N
          WRITE (13,220) I,X(I),I,Y(I),I,T(I)
220      FORMAT ('X',I2,1X,'=',D17.10,5X,'Y',I2,1X,'=',D17.10,5
          'T',I2,1X,'=',D17.10)
200      CONTINUE
      RETURN
      END

```

Figure 3.10 Fortran Program of FIR Filter in Shift/Add Form

input, function (the filter), and output, the program was easier to write and easier to understand. Second, it allowed the section to be later implemented in the filter hardware to be separated from the rest of the program. Finally, it allowed changes to be made easily to the input and output routines during the many phases of development and will be useful for any follow-on work that might be done with this filter.

After completing and running this version of the filter implementation, the results were found to be the same as the rational polynomial form of the filter. In fact, these equations were transferred to the "DSL" program as shown in Figure 3.11 and the graphs produced were identical to the rational polynomial graphs shown earlier. The problem encountered in running the filter in Fortran on the VAX was that although a stream of output data was produced, there was not the quick visual reference as provided by the "DSL" program. The Fortran program proved to be useful later on however, when Root Mean Square (RMS) values were desired and also when flags were added to the program to determine overflow conditions as will be shown.

#### D. FIXED POINT IMPLEMENTATION AND QUANTIZATION NOISE EFFECTS

The next and final step before being able to implement the filter at the bit-slice level was to implement the

```

TITLE DIGITAL FILTER(REAL TIME RESPONSE)
INITIAL Y=0.
INITIAL X1=0.,X2=0.,Y14=0.,Y13=0.,Y12=0.,Y11=0.,Y23=0.,Y22=0.,Y21=0.
INITIAL Y31=0.,Y41=0.,Y52=0.,Y51=0.
INITIAL X=0.
CONST B=1.0
CONST F=3.58E6
CONST FS=1.432E7
DYNAMIC
    TIME1=K/FS
    THETA=2.*PI*F*TIME1
    X=B*SIN(THETA)
    Y1=X-X2
    X2=X1
    X1=X
    Y2=Y1+Y14
    Y14=Y13
    Y13=Y12
    Y12=Y11
    Y11=Y1
    Y3=Y2+Y23
    Y23=Y22
    Y22=Y21
    Y21=Y2
    Y4=Y3-Y31
    Y31=Y3
    Y5=Y4-Y41
    Y41=Y4
    Y=Y5-Y52
    Y52=Y51
    Y51=Y5
RENAME TIME=K
CONTROL FINTIM=100,DELT=1.
PRINT 1.,TIME1,X,Y
SAVE 1.,TIME1,X,Y
GRAPH (DE=TEK618) TIME1(UN=SECS),Y(MA=5)
GRAPH (DE=TEK618) TIME1(UN=SECS),X(MA=4)
LABEL OUTPUT OF DIGITAL FILTER
LABEL INPUT TO DIGITAL FILTER
END
STOP

```

Figure 3.11 DSL Program of FIR Filter in Shift/Add Form

filter using fixed point precision arithmetic and to observe the effects of truncation noise introduced. Although the 29203 evaluation board allowed for 16 bit precision in its ALU processor, Dr. Lee imposed the additional constraint of implementing the filter in bit-slice using only 8 of the 16 bits on the 29203 evaluation board. The purpose for this change is to allow for easier implementation in discrete random logic hardware at a later date.

Up to this point, the computer was assumed to have infinite precision with no effects due to converting from an analog signal to a digital signal through sampling. This conversion from a smooth curve in the ideal case to a signal which has been restricted to a fixed number of signal levels or quantization levels in the sampled case introduces what is known as quantization noise. In the floating point case the precision is assumed infinite, but when comparing the RMS value of the floating point 100KHz  $10 \cdot \sin(\Theta)$  input signal to the ideal RMS value (0.707 of the magnitude), the error is found to be 1.087 or approximately 15%. (Note: The RMS values were obtained by inserting instructions in the Fortran program to add the squared sampled values over the period of a complete sine wave, taking the average of the sum and then taking the square root to obtain the RMS value.) When comparing the RMS value of a fixed point 100KHz  $10 \cdot \sin(\Theta)$  input signal to that of the ideal signal, the error was found to be 1.531, a difference of

only 0.45 from the floating point case. This difference of approximately 0.4 did not change significantly as the magnitude of the input signal was increased. Although this difference did not appear to be significant, the difference between the floating point and fixed point signal input had a significant effect on the output of the filter as will be shown in the next paragraph. This data is presented in Table I and is summarized below.

The major concentration of effort was spent in looking at how the fixed point output of the filter differed from the floating point case. For the out-of-band floating point signal,  $10 \cdot \sin(\Theta)$  at 100 KHz, the steady state RMS value was found to be very nearly zero at  $0.391 \times 10^{-3}$  which is shown to be negligible in Figure 3.12. For the same signal in fixed point, the noise is found to be significant, with the steady state signal ranging in value from -9 to +9 and with an RMS value of 3.86 as shown in Figure 3.13. The noise which is introduced is first due to the limited number of fixed point quantization levels, with the signal ranging from -10 to +10 in increments of 1, which also causes the sampled signal to be truncated. To be exact, the signal ranges from -9 to +9 due to the fact that the computer truncates the signal to the next lower number in the positive case and to the next higher number in the negative case. It is this truncation of the signal which introduces

TABLE I

## RMS INPUT AND OUTPUT VALUES

INPUT FUNCTION	10*SINθ	63*SINθ	127*SINθ	1270*SINθ	12700*SINθ
SAMPLED RMS INPUT FLOATING PT	5.983	37.69	75.99	759.88	7598.87
SAMPLED RMS INPUT FIXED PT	5.539	37.28	75.59	759.50	7598.51
IDEAL RMS INPUT	7.07	44.54	89.79	897.89	8978.90
DIFFERENCE BETWEEN FLOATING & FIXED	.45	.41	.41	.38	.36
DIFFERENCE BETWEEN FLOATING & IDEAL	15.4%	15.4%	15.4%	15.4%	15.4%
FLOATING POINT RMS OUTPUT	.391E-3	.246E-2	.496E-2	.496E-1	.496
FIXED POINT RMS OUTPUT	4.6	3.3	3.5	5.4	4.9

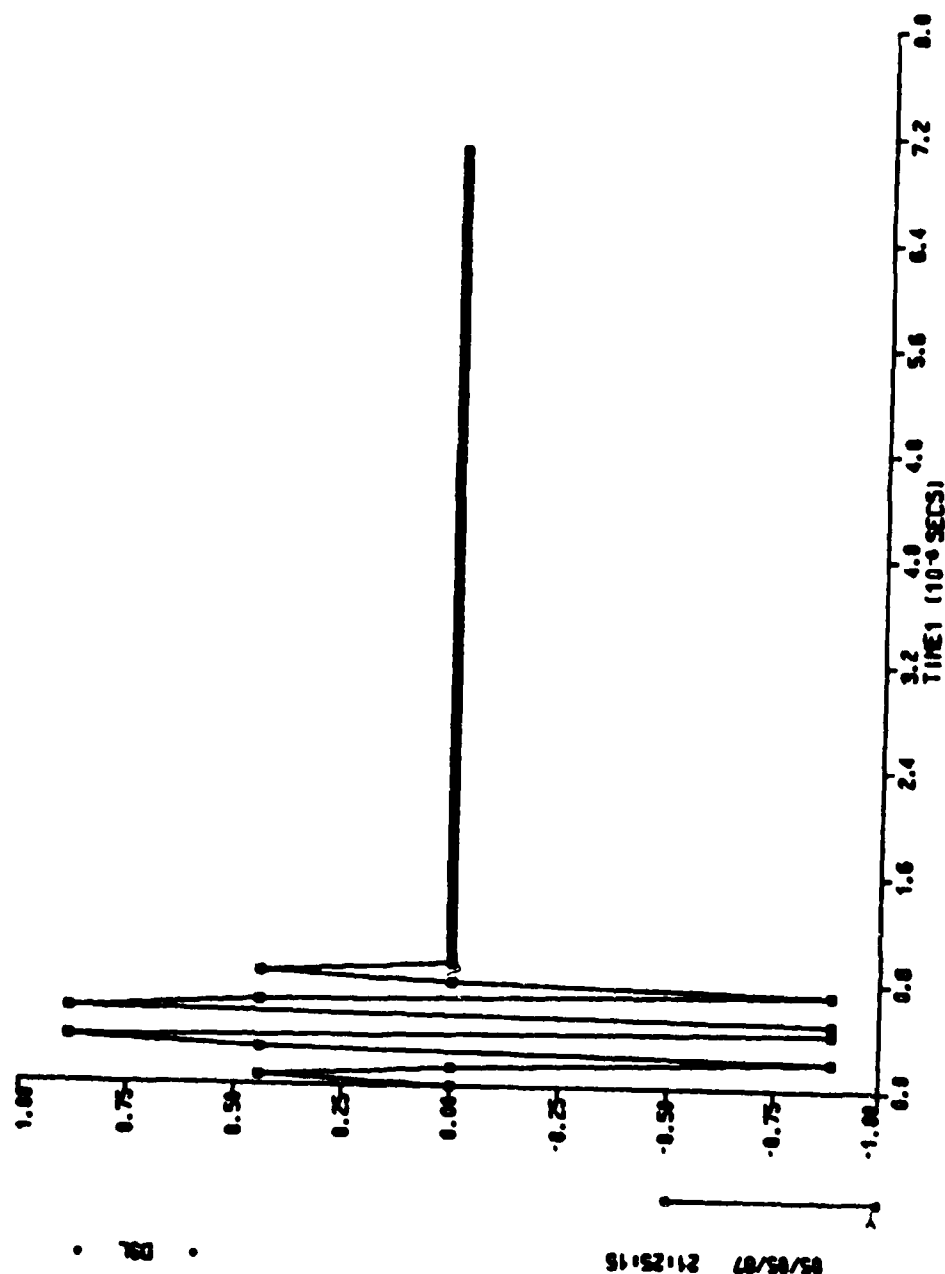


Figure 3.12 DSL Graph of Floating Point Filtering With Input of  $10 \cdot \sin(\theta)$  at 100KHz

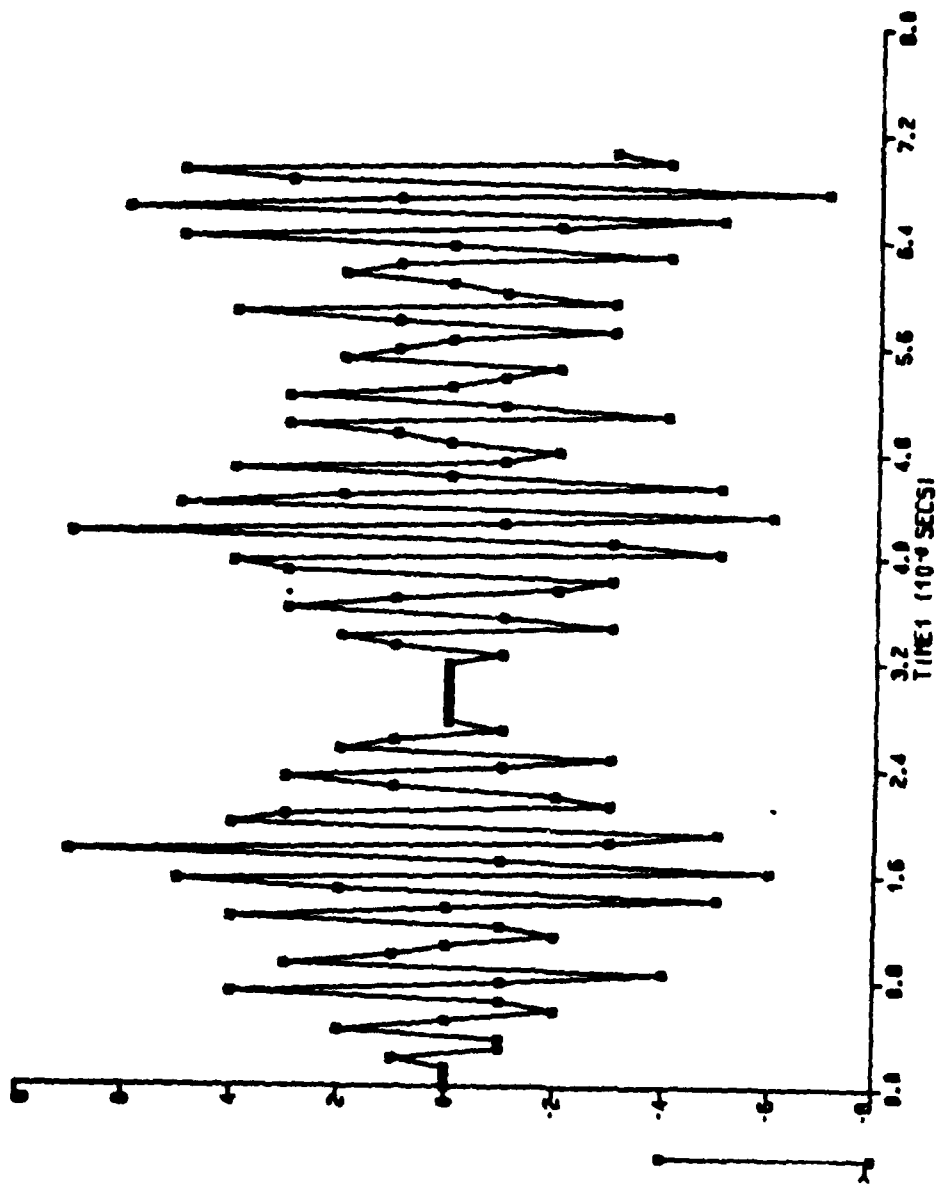


Figure 3.13 DSL Graph of Fixed Point Filtering with Input  
of 10\*Sin(Theta) at 100KHz

• 00 •

05/05/87 21:27:15



additional noise. As the signal was increased in value, for the floating point case, the signal-to-noise ratio remained constant. That is to say, for every 10-fold increase in the signal level, the output noise level was also increased by 10-fold. Although this is not an analog signal, this seemed to correlate with the statement made by Gold [Ref. 14] that every analog signal will have some finite signal-to-noise ratio. Therefore, increasing the accuracy by which the signal is represented will only increase the accuracy by which the noise is represented as well. In the fixed point case, however, as the accuracy of the signal representation was increased, the output noise level remained fairly constant with an RMS value ranging approximately between 3 and 4. As shown in Figures 3.14 and 3.15, this noise becomes less and less significant as the input signal is increased and therefore the signal-to-noise ratio is also increased.

With this information in hand, the next problem was to determine the maximum signal which could be used as an input to the filter without producing an overflow, using the available 8 bits of accuracy. Using 8 bits, the integer signal levels could range from -128 to +127 in the two's complement representation. This meant however, that with the gain of 23 produced by the filter at the subcarrier frequency of 3.58 MHz, the maximum input to the filter would be approximately  $5 \cdot \sin(\Theta)$ . For the maximum gain of the

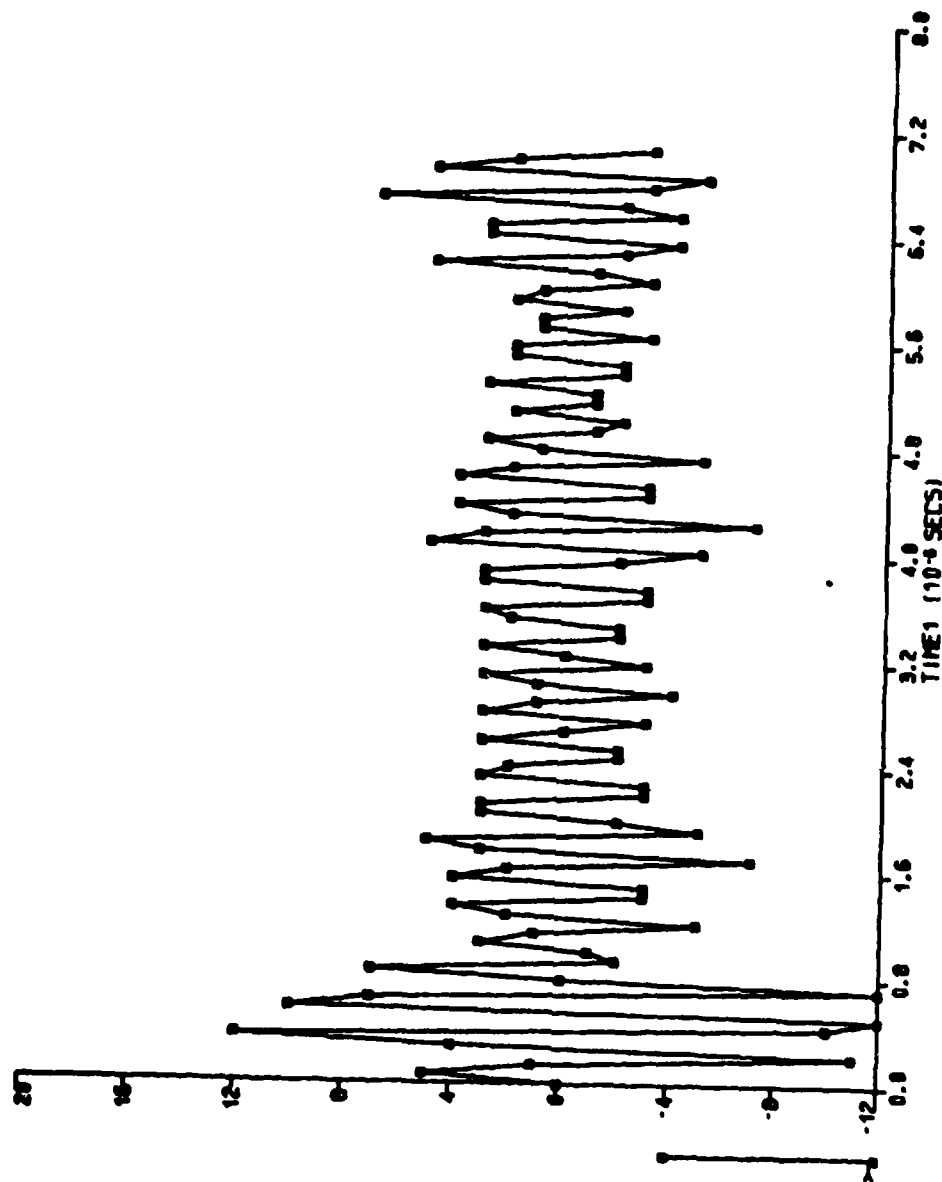


Figure 3.14 DSL Graph of Fixed Point Filtering with Input of  $127 \cdot \sin(\theta)$  at 100KHz

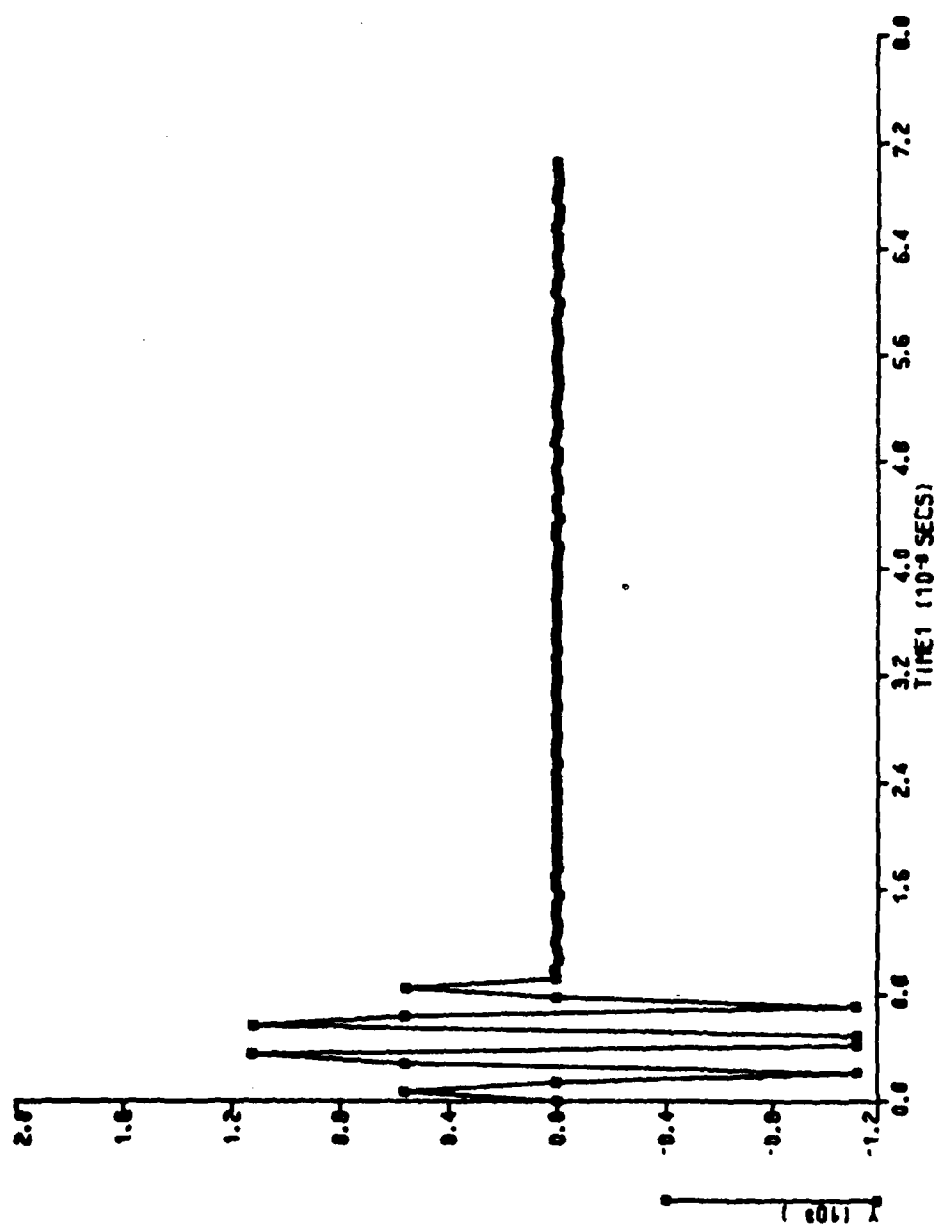


Figure 3.15 DSL Graph of Fixed Point Filtering with Input  
of  $12700 \cdot \sin(I\theta)$  at 100KHz

filter of approximately 30, the maximum input signal would have to be even less. As seen by the previous discussion, this would not provide the necessary accuracy needed in quantization levels of the signal. To compensate for this, the signal was divided by two after each adder in the filter, as shown in the "DSL" program of Figure 3.16. Dividing by two allowed implementation at the bit-slice level using shifters rather than expensive and time-consuming dividers. Now with these dividers in place, the maximum input signal to the filter as well as the number of dividers actually required needed to be determined. To accomplish this, the Fortran version of the program was used and a flag was inserted after each adder to determine if an overflow condition existed with a given input magnitude. The magnitude was incremented in steps through the use of a DO LOOP in the main calling program. This program is shown in Figure 3.17. It had appeared, using "DSL", that a maximum signal of  $127 \cdot \sin(\text{THETA})$  could be used with 5 of the 6 dividers in place to produce an output signal of approximately  $91 \cdot \sin(\text{THETA})$  without producing an overflow condition as shown in Figure 3.18. However, using the flag program on the VAX Fortran, it was found that the first adder limited the input signal to a magnitude of  $63 \cdot \sin(\text{THETA})$ . Anything above this magnitude would cause an overflow condition to occur. This resulted in an output magnitude of only  $45 \cdot \sin(\text{THETA})$  which meant that the 8 bit

```

TITLE DIGITAL FILTER(REAL TIME RESPONSE)
FIXED X,Y,X1,X2,Y14,Y13,Y12,Y11,Y23,Y22,Y21,Y31,Y41,Y52,Y51
FIXED Y1,Y2,Y3,Y4,Y5
INITIAL Y=0
INITIAL X1=0,X2=0,Y14=0,Y13=0,Y12=0,Y11=0,Y23=0,Y22=0,Y21=0
INITIAL Y31=0,Y41=0,Y52=0,Y51=0
INITIAL X=0
CONST B=63.000
CONST F=1.E5
CONST FS=14200000.0
DYNAMIC
    TIME1=K/FS
    THETA=2.*PI*F*TIME1
    XX=B*SIN( THETA)
    X=INT( XX)
    Y1=X-X2
    Y1=Y1/2
    X2=X1
    X1=X
    Y2=Y1+Y14
    Y2=Y2/2
    Y14=Y13
    Y13=Y12
    Y12=Y11
    Y11=Y1
    Y3=Y2+Y23
    Y3=Y3/2
    Y23=Y22
    Y22=Y21
    Y21=Y2
    Y4=Y3-Y31
    Y4=Y4/2
    Y31=Y3
    Y5=Y4-Y41
    Y41=Y4
    Y=Y5-Y52
    Y52=Y51
    Y51=Y5
RENAME TIME=K
CONTROL FINTIM=100,DELT=1.
PRINT 1.,TIME1,X,Y
SAVE 1.,TIME1,X,Y
GRAPH (DE=TEK618) TIME1(UN=SECS),Y(MA=5)
GRAPH (DE=TEK618) TIME1(UN=SECS),X(MA=4)
LABEL OUTPUT OF DIGITAL FILTER
LABEL INPUT TO DIGITAL FILTER
END
STOP

```

Figure 3.16 DSL FIR Filter Program in Shift/Add Form With Dividers Added

```

C      THIS PROGRAM IS A REPRESENTATION OF A 13TH ORDER BAND PASS FILTER
C
      INTEGER X(100),Y(100)
      REAL Y(101)
      INTEGER N,IMOV,B,KF,B,SINCA
      PRINT 6
      FORMAT ('1')
      DO 40 SINCA=40,70
        B=SINCA
        CALL INPUT (N,X,T,B)
        CALL FUNCT (X,T,N,IMOV,B,KF)
        CALL OUTPUT (X,T,B,KF,IMOV)
        PRINT 6
        IF (IMOV.GT.0) THEN
          GO TO 10
        END IF
      40 CONTINUE
      STOP
      END
-----
      SUBROUTINE INPUT (N,X,TIME1,B)
      REAL XM(100),F,FS,TIME1(101),THETA
      INTEGER N(100)
      INTEGER N,K,B
      M=100
      F=1.5866
      FS=1.432E7
      TIME1(1)=0.
      DO 100 K=1,N
        THETA=3.1415926*F*TIME1(K)
        XM(K)=FLOAT(B)*SIN(THETA)
        X(K)=INT(XM(K))
        TIME1(K+1)=X/FS
      100 CONTINUE
      RETURN
      END
-----
      SUBROUTINE FUNCT (X,T,N,IMOV,B,KF)
      INTEGER X(100),Y(100),Y1,Y2,Y3,Y4,Y5
      INTEGER X1/X2/X3/X4/X5/Y1/Y2/Y3/Y4/Y5/Y6/Y7/Y8/Y9/Y10/Y11/Y12/Y13/Y14/Y15/Y16/Y17/Y18/Y19/Y20/Y21/Y22/Y23/Y24/Y25/Y26/Y27/Y28/Y29/Y30/Y31/Y32/Y33/Y34/Y35/Y36/Y37/Y38/Y39/Y40/Y41/Y42/Y43/Y44/Y45/Y46/Y47/Y48/Y49/Y50/Y51/Y52/Y53/Y54/Y55/Y56/Y57/Y58/Y59/Y60/Y61/Y62/Y63/Y64/Y65/Y66/Y67/Y68/Y69/Y70/Y71/Y72/Y73/Y74/Y75/Y76/Y77/Y78/Y79/Y80/Y81/Y82/Y83/Y84/Y85/Y86/Y87/Y88/Y89/Y90/Y91/Y92/Y93/Y94/Y95/Y96/Y97/Y98/Y99/Y100
      INTEGER Y1/Y2/Y3/Y4/Y5/Y6/Y7/Y8/Y9/Y10/Y11/Y12/Y13/Y14/Y15/Y16/Y17/Y18/Y19/Y20/Y21/Y22/Y23/Y24/Y25/Y26/Y27/Y28/Y29/Y30/Y31/Y32/Y33/Y34/Y35/Y36/Y37/Y38/Y39/Y40/Y41/Y42/Y43/Y44/Y45/Y46/Y47/Y48/Y49/Y50/Y51/Y52/Y53/Y54/Y55/Y56/Y57/Y58/Y59/Y60/Y61/Y62/Y63/Y64/Y65/Y66/Y67/Y68/Y69/Y70/Y71/Y72/Y73/Y74/Y75/Y76/Y77/Y78/Y79/Y80/Y81/Y82/Y83/Y84/Y85/Y86/Y87/Y88/Y89/Y90/Y91/Y92/Y93/Y94/Y95/Y96/Y97/Y98/Y99/Y100
      IMOV=B
      DO 60 KF=1,N
        Y1=X(KF)-X3
        IFLAG=1
        CALL OVERFLO (Y1,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        CALL SHIFT (Y1)
        Y1=Y1/2
        X3=X1
        X1=X(KF)
        Y2=Y1-Y1/4
        IFLAG=2
        CALL OVERFLO (Y2,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        CALL SHIFT (Y2)
        Y1=Y1+Y2
        Y2=Y1
        Y3=Y1-Y2
        IFLAG=3
        CALL OVERFLO (Y3,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        CALL SHIFT (Y3)
        Y2=Y2+Y3
        Y3=Y2
        Y4=Y2-Y3
        IFLAG=4
        CALL OVERFLO (Y4,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        CALL SHIFT (Y4)
        Y3=Y3+Y4
        Y4=Y3
        Y5=Y3-Y4
        IFLAG=5
        CALL OVERFLO (Y5,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        CALL SHIFT (Y5)
        Y4=Y4+Y5
        Y5=Y4
        Y6=Y4-Y5
        IFLAG=6
        CALL OVERFLO (Y6,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y5=Y5+Y6
        Y6=Y5
        Y7=Y5-Y6
        IFLAG=7
        CALL OVERFLO (Y7,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y6=Y6+Y7
        Y7=Y6
        Y8=Y6-Y7
        IFLAG=8
        CALL OVERFLO (Y8,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y7=Y7+Y8
        Y8=Y7
        Y9=Y7-Y8
        IFLAG=9
        CALL OVERFLO (Y9,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y8=Y8+Y9
        Y9=Y8
        Y10=Y8-Y9
        IFLAG=10
        CALL OVERFLO (Y10,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y9=Y9+Y10
        Y10=Y9
        Y11=Y9-Y10
        IFLAG=11
        CALL OVERFLO (Y11,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y10=Y10+Y11
        Y11=Y10
        Y12=Y10-Y11
        IFLAG=12
        CALL OVERFLO (Y12,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y11=Y11+Y12
        Y12=Y11
        Y13=Y11-Y12
        IFLAG=13
        CALL OVERFLO (Y13,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y12=Y12+Y13
        Y13=Y12
        Y14=Y12-Y13
        IFLAG=14
        CALL OVERFLO (Y14,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y13=Y13+Y14
        Y14=Y13
        Y15=Y13-Y14
        IFLAG=15
        CALL OVERFLO (Y15,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y14=Y14+Y15
        Y15=Y14
        Y16=Y14-Y15
        IFLAG=16
        CALL OVERFLO (Y16,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y15=Y15+Y16
        Y16=Y15
        Y17=Y15-Y16
        IFLAG=17
        CALL OVERFLO (Y17,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y16=Y16+Y17
        Y17=Y16
        Y18=Y16-Y17
        IFLAG=18
        CALL OVERFLO (Y18,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y17=Y17+Y18
        Y18=Y17
        Y19=Y17-Y18
        IFLAG=19
        CALL OVERFLO (Y19,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y18=Y18+Y19
        Y19=Y18
        Y20=Y18-Y19
        IFLAG=20
        CALL OVERFLO (Y20,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y19=Y19+Y20
        Y20=Y19
        Y21=Y19-Y20
        IFLAG=21
        CALL OVERFLO (Y21,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y20=Y20+Y21
        Y21=Y20
        Y22=Y20-Y21
        IFLAG=22
        CALL OVERFLO (Y22,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y21=Y21+Y22
        Y22=Y21
        Y23=Y21-Y22
        IFLAG=23
        CALL OVERFLO (Y23,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y22=Y22+Y23
        Y23=Y22
        Y24=Y22-Y23
        IFLAG=24
        CALL OVERFLO (Y24,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y23=Y23+Y24
        Y24=Y23
        Y25=Y23-Y24
        IFLAG=25
        CALL OVERFLO (Y25,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y24=Y24+Y25
        Y25=Y24
        Y26=Y24-Y25
        IFLAG=26
        CALL OVERFLO (Y26,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y25=Y25+Y26
        Y26=Y25
        Y27=Y25-Y26
        IFLAG=27
        CALL OVERFLO (Y27,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y26=Y26+Y27
        Y27=Y26
        Y28=Y26-Y27
        IFLAG=28
        CALL OVERFLO (Y28,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y27=Y27+Y28
        Y28=Y27
        Y29=Y27-Y28
        IFLAG=29
        CALL OVERFLO (Y29,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y28=Y28+Y29
        Y29=Y28
        Y30=Y28-Y29
        IFLAG=30
        CALL OVERFLO (Y30,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y29=Y29+Y30
        Y30=Y29
        Y31=Y29-Y30
        IFLAG=31
        CALL OVERFLO (Y31,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y30=Y30+Y31
        Y31=Y30
        Y32=Y30-Y31
        IFLAG=32
        CALL OVERFLO (Y32,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y31=Y31+Y32
        Y32=Y31
        Y33=Y31-Y32
        IFLAG=33
        CALL OVERFLO (Y33,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y32=Y32+Y33
        Y33=Y32
        Y34=Y32-Y33
        IFLAG=34
        CALL OVERFLO (Y34,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y33=Y33+Y34
        Y34=Y33
        Y35=Y33-Y34
        IFLAG=35
        CALL OVERFLO (Y35,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y34=Y34+Y35
        Y35=Y34
        Y36=Y34-Y35
        IFLAG=36
        CALL OVERFLO (Y36,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y35=Y35+Y36
        Y36=Y35
        Y37=Y35-Y36
        IFLAG=37
        CALL OVERFLO (Y37,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y36=Y36+Y37
        Y37=Y36
        Y38=Y36-Y37
        IFLAG=38
        CALL OVERFLO (Y38,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y37=Y37+Y38
        Y38=Y37
        Y39=Y37-Y38
        IFLAG=39
        CALL OVERFLO (Y39,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y38=Y38+Y39
        Y39=Y38
        Y40=Y38-Y39
        IFLAG=40
        CALL OVERFLO (Y40,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y39=Y39+Y40
        Y40=Y39
        Y41=Y39-Y40
        IFLAG=41
        CALL OVERFLO (Y41,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y40=Y40+Y41
        Y41=Y40
        Y42=Y40-Y41
        IFLAG=42
        CALL OVERFLO (Y42,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y41=Y41+Y42
        Y42=Y41
        Y43=Y41-Y42
        IFLAG=43
        CALL OVERFLO (Y43,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y42=Y42+Y43
        Y43=Y42
        Y44=Y42-Y43
        IFLAG=44
        CALL OVERFLO (Y44,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y43=Y43+Y44
        Y44=Y43
        Y45=Y43-Y44
        IFLAG=45
        CALL OVERFLO (Y45,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y44=Y44+Y45
        Y45=Y44
        Y46=Y44-Y45
        IFLAG=46
        CALL OVERFLO (Y46,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y45=Y45+Y46
        Y46=Y45
        Y47=Y45-Y46
        IFLAG=47
        CALL OVERFLO (Y47,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y46=Y46+Y47
        Y47=Y46
        Y48=Y46-Y47
        IFLAG=48
        CALL OVERFLO (Y48,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y47=Y47+Y48
        Y48=Y47
        Y49=Y47-Y48
        IFLAG=49
        CALL OVERFLO (Y49,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y48=Y48+Y49
        Y49=Y48
        Y50=Y48-Y49
        IFLAG=50
        CALL OVERFLO (Y50,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y49=Y49+Y50
        Y50=Y49
        Y51=Y49-Y50
        IFLAG=51
        CALL OVERFLO (Y51,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y50=Y50+Y51
        Y51=Y50
        Y52=Y50-Y51
        IFLAG=52
        CALL OVERFLO (Y52,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y51=Y51+Y52
        Y52=Y51
        Y53=Y51-Y52
        IFLAG=53
        CALL OVERFLO (Y53,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y52=Y52+Y53
        Y53=Y52
        Y54=Y52-Y53
        IFLAG=54
        CALL OVERFLO (Y54,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y53=Y53+Y54
        Y54=Y53
        Y55=Y53-Y54
        IFLAG=55
        CALL OVERFLO (Y55,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y54=Y54+Y55
        Y55=Y54
        Y56=Y54-Y55
        IFLAG=56
        CALL OVERFLO (Y56,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y55=Y55+Y56
        Y56=Y55
        Y57=Y55-Y56
        IFLAG=57
        CALL OVERFLO (Y57,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y56=Y56+Y57
        Y57=Y56
        Y58=Y56-Y57
        IFLAG=58
        CALL OVERFLO (Y58,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y57=Y57+Y58
        Y58=Y57
        Y59=Y57-Y58
        IFLAG=59
        CALL OVERFLO (Y59,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y58=Y58+Y59
        Y59=Y58
        Y60=Y58-Y59
        IFLAG=60
        CALL OVERFLO (Y60,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y59=Y59+Y60
        Y60=Y59
        Y61=Y59-Y60
        IFLAG=61
        CALL OVERFLO (Y61,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y60=Y60+Y61
        Y61=Y60
        Y62=Y60-Y61
        IFLAG=62
        CALL OVERFLO (Y62,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y61=Y61+Y62
        Y62=Y61
        Y63=Y61-Y62
        IFLAG=63
        CALL OVERFLO (Y63,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y62=Y62+Y63
        Y63=Y62
        Y64=Y62-Y63
        IFLAG=64
        CALL OVERFLO (Y64,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y63=Y63+Y64
        Y64=Y63
        Y65=Y63-Y64
        IFLAG=65
        CALL OVERFLO (Y65,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y64=Y64+Y65
        Y65=Y64
        Y66=Y64-Y65
        IFLAG=66
        CALL OVERFLO (Y66,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y65=Y65+Y66
        Y66=Y65
        Y67=Y65-Y66
        IFLAG=67
        CALL OVERFLO (Y67,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y66=Y66+Y67
        Y67=Y66
        Y68=Y66-Y67
        IFLAG=68
        CALL OVERFLO (Y68,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y67=Y67+Y68
        Y68=Y67
        Y69=Y67-Y68
        IFLAG=69
        CALL OVERFLO (Y69,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y68=Y68+Y69
        Y69=Y68
        Y70=Y68-Y69
        IFLAG=70
        CALL OVERFLO (Y70,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y69=Y69+Y70
        Y70=Y69
        Y71=Y69-Y70
        IFLAG=71
        CALL OVERFLO (Y71,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y70=Y70+Y71
        Y71=Y70
        Y72=Y70-Y71
        IFLAG=72
        CALL OVERFLO (Y72,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y71=Y71+Y72
        Y72=Y71
        Y73=Y71-Y72
        IFLAG=73
        CALL OVERFLO (Y73,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y72=Y72+Y73
        Y73=Y72
        Y74=Y72-Y73
        IFLAG=74
        CALL OVERFLO (Y74,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y73=Y73+Y74
        Y74=Y73
        Y75=Y73-Y74
        IFLAG=75
        CALL OVERFLO (Y75,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y74=Y74+Y75
        Y75=Y74
        Y76=Y74-Y75
        IFLAG=76
        CALL OVERFLO (Y76,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y75=Y75+Y76
        Y76=Y75
        Y77=Y75-Y76
        IFLAG=77
        CALL OVERFLO (Y77,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y76=Y76+Y77
        Y77=Y76
        Y78=Y76-Y77
        IFLAG=78
        CALL OVERFLO (Y78,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y77=Y77+Y78
        Y78=Y77
        Y79=Y77-Y78
        IFLAG=79
        CALL OVERFLO (Y79,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y78=Y78+Y79
        Y79=Y78
        Y80=Y78-Y79
        IFLAG=80
        CALL OVERFLO (Y80,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y79=Y79+Y80
        Y80=Y79
        Y81=Y79-Y80
        IFLAG=81
        CALL OVERFLO (Y81,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y80=Y80+Y81
        Y81=Y80
        Y82=Y80-Y81
        IFLAG=82
        CALL OVERFLO (Y82,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y81=Y81+Y82
        Y82=Y81
        Y83=Y81-Y82
        IFLAG=83
        CALL OVERFLO (Y83,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y82=Y82+Y83
        Y83=Y82
        Y84=Y82-Y83
        IFLAG=84
        CALL OVERFLO (Y84,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y83=Y83+Y84
        Y84=Y83
        Y85=Y83-Y84
        IFLAG=85
        CALL OVERFLO (Y85,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y84=Y84+Y85
        Y85=Y84
        Y86=Y84-Y85
        IFLAG=86
        CALL OVERFLO (Y86,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y85=Y85+Y86
        Y86=Y85
        Y87=Y85-Y86
        IFLAG=87
        CALL OVERFLO (Y87,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y86=Y86+Y87
        Y87=Y86
        Y88=Y86-Y87
        IFLAG=88
        CALL OVERFLO (Y88,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y87=Y87+Y88
        Y88=Y87
        Y89=Y87-Y88
        IFLAG=89
        CALL OVERFLO (Y89,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y88=Y88+Y89
        Y89=Y88
        Y90=Y88-Y89
        IFLAG=90
        CALL OVERFLO (Y90,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y89=Y89+Y90
        Y90=Y89
        Y91=Y89-Y90
        IFLAG=91
        CALL OVERFLO (Y91,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y90=Y90+Y91
        Y91=Y90
        Y92=Y90-Y91
        IFLAG=92
        CALL OVERFLO (Y92,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y91=Y91+Y92
        Y92=Y91
        Y93=Y91-Y92
        IFLAG=93
        CALL OVERFLO (Y93,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y92=Y92+Y93
        Y93=Y92
        Y94=Y92-Y93
        IFLAG=94
        CALL OVERFLO (Y94,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y93=Y93+Y94
        Y94=Y93
        Y95=Y93-Y94
        IFLAG=95
        CALL OVERFLO (Y95,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y94=Y94+Y95
        Y95=Y94
        Y96=Y94-Y95
        IFLAG=96
        CALL OVERFLO (Y96,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y95=Y95+Y96
        Y96=Y95
        Y97=Y95-Y96
        IFLAG=97
        CALL OVERFLO (Y97,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y96=Y96+Y97
        Y97=Y96
        Y98=Y96-Y97
        IFLAG=98
        CALL OVERFLO (Y98,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y97=Y97+Y98
        Y98=Y97
        Y99=Y97-Y98
        IFLAG=99
        CALL OVERFLO (Y99,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y98=Y98+Y99
        Y99=Y98
        Y100=Y98-Y99
        IFLAG=100
        CALL OVERFLO (Y100,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y99=Y99+Y100
        Y100=Y99
        Y101=Y99-Y100
        IFLAG=101
        CALL OVERFLO (Y101,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y100=Y100+Y101
        Y101=Y100
        Y102=Y100-Y101
        IFLAG=102
        CALL OVERFLO (Y102,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y101=Y101+Y102
        Y102=Y101
        Y103=Y101-Y102
        IFLAG=103
        CALL OVERFLO (Y103,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y102=Y102+Y103
        Y103=Y102
        Y104=Y102-Y103
        IFLAG=104
        CALL OVERFLO (Y104,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y103=Y103+Y104
        Y104=Y103
        Y105=Y103-Y104
        IFLAG=105
        CALL OVERFLO (Y105,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y104=Y104+Y105
        Y105=Y104
        Y106=Y104-Y105
        IFLAG=106
        CALL OVERFLO (Y106,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y105=Y105+Y106
        Y106=Y105
        Y107=Y105-Y106
        IFLAG=107
        CALL OVERFLO (Y107,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y106=Y106+Y107
        Y107=Y106
        Y108=Y106-Y107
        IFLAG=108
        CALL OVERFLO (Y108,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y107=Y107+Y108
        Y108=Y107
        Y109=Y107-Y108
        IFLAG=109
        CALL OVERFLO (Y109,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y108=Y108+Y109
        Y109=Y108
        Y110=Y108-Y109
        IFLAG=110
        CALL OVERFLO (Y110,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y109=Y109+Y110
        Y110=Y109
        Y111=Y109-Y110
        IFLAG=111
        CALL OVERFLO (Y111,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y110=Y110+Y111
        Y111=Y110
        Y112=Y110-Y111
        IFLAG=112
        CALL OVERFLO (Y112,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y111=Y111+Y112
        Y112=Y111
        Y113=Y111-Y112
        IFLAG=113
        CALL OVERFLO (Y113,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y112=Y112+Y113
        Y113=Y112
        Y114=Y112-Y113
        IFLAG=114
        CALL OVERFLO (Y114,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y113=Y113+Y114
        Y114=Y113
        Y115=Y113-Y114
        IFLAG=115
        CALL OVERFLO (Y115,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y114=Y114+Y115
        Y115=Y114
        Y116=Y114-Y115
        IFLAG=116
        CALL OVERFLO (Y116,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y115=Y115+Y116
        Y116=Y115
        Y117=Y115-Y116
        IFLAG=117
        CALL OVERFLO (Y117,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y116=Y116+Y117
        Y117=Y116
        Y118=Y116-Y117
        IFLAG=118
        CALL OVERFLO (Y118,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y117=Y117+Y118
        Y118=Y117
        Y119=Y117-Y118
        IFLAG=119
        CALL OVERFLO (Y119,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y118=Y118+Y119
        Y119=Y118
        Y120=Y118-Y119
        IFLAG=120
        CALL OVERFLO (Y120,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y119=Y119+Y120
        Y120=Y119
        Y121=Y119-Y120
        IFLAG=121
        CALL OVERFLO (Y121,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y120=Y120+Y121
        Y121=Y120
        Y122=Y120-Y121
        IFLAG=122
        CALL OVERFLO (Y122,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y121=Y121+Y122
        Y122=Y121
        Y123=Y121-Y122
        IFLAG=123
        CALL OVERFLO (Y123,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y122=Y122+Y123
        Y123=Y122
        Y124=Y122-Y123
        IFLAG=124
        CALL OVERFLO (Y124,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y123=Y123+Y124
        Y124=Y123
        Y125=Y123-Y124
        IFLAG=125
        CALL OVERFLO (Y125,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y124=Y124+Y125
        Y125=Y124
        Y126=Y124-Y125
        IFLAG=126
        CALL OVERFLO (Y126,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y125=Y125+Y126
        Y126=Y125
        Y127=Y125-Y126
        IFLAG=127
        CALL OVERFLO (Y127,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y126=Y126+Y127
        Y127=Y126
        Y128=Y126-Y127
        IFLAG=128
        CALL OVERFLO (Y128,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y127=Y127+Y128
        Y128=Y127
        Y129=Y127-Y128
        IFLAG=129
        CALL OVERFLO (Y129,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y128=Y128+Y129
        Y129=Y128
        Y130=Y128-Y129
        IFLAG=130
        CALL OVERFLO (Y130,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y129=Y129+Y130
        Y130=Y129
        Y131=Y129-Y130
        IFLAG=131
        CALL OVERFLO (Y131,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y130=Y130+Y131
        Y131=Y130
        Y132=Y130-Y131
        IFLAG=132
        CALL OVERFLO (Y132,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y131=Y131+Y132
        Y132=Y131
        Y133=Y131-Y132
        IFLAG=133
        CALL OVERFLO (Y133,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y132=Y132+Y133
        Y133=Y132
        Y134=Y132-Y133
        IFLAG=134
        CALL OVERFLO (Y134,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y133=Y133+Y134
        Y134=Y133
        Y135=Y133-Y134
        IFLAG=135
        CALL OVERFLO (Y135,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y134=Y134+Y135
        Y135=Y134
        Y136=Y134-Y135
        IFLAG=136
        CALL OVERFLO (Y136,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y135=Y135+Y136
        Y136=Y135
        Y137=Y135-Y136
        IFLAG=137
        CALL OVERFLO (Y137,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y136=Y136+Y137
        Y137=Y136
        Y138=Y136-Y137
        IFLAG=138
        CALL OVERFLO (Y138,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y137=Y137+Y138
        Y138=Y137
        Y139=Y137-Y138
        IFLAG=139
        CALL OVERFLO (Y139,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y138=Y138+Y139
        Y139=Y138
        Y140=Y138-Y139
        IFLAG=140
        CALL OVERFLO (Y140,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y139=Y139+Y140
        Y140=Y139
        Y141=Y139-Y140
        IFLAG=141
        CALL OVERFLO (Y141,IMOV,IFLAG)
        IF (IMOV.GT.0) THEN
          GO TO 60
        END IF
        Y140=Y140+Y14
```

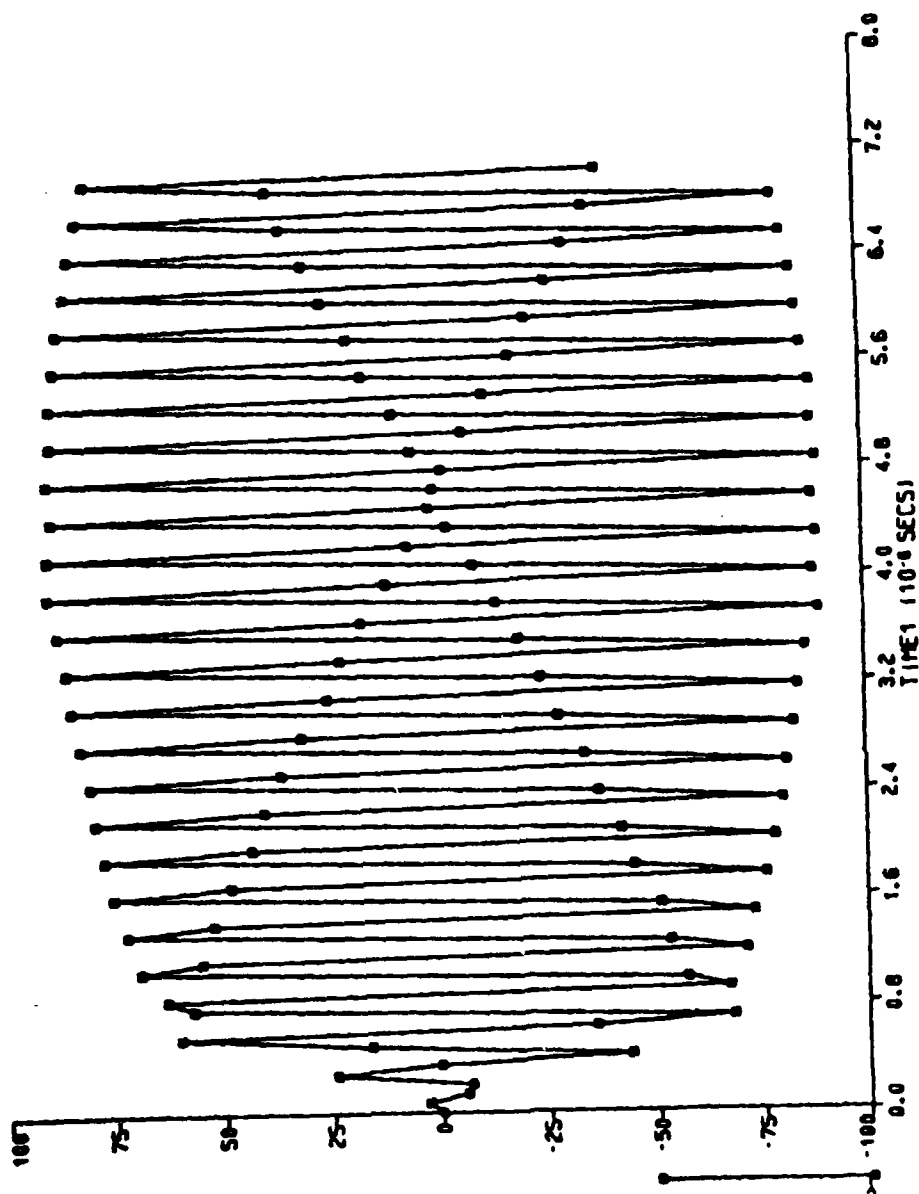


Figure 3.18 DSL Graph of Fixed Point Filtering of Sine Wave  
With Filter in Shift/Add Form with Five  
Dividers Added

• DSL •

04/28/87 15:53:30

accuracy of the bit-slice would not be fully utilized. The dividers were then removed one at a time in a progressive manner through the filter and it was determined that with the limiting input magnitude of  $63 \cdot \sin(\text{THETA})$ , the fifth divider was no longer needed in the filter. This meant that the maximum output magnitude of the filter at the subcarrier frequency of 3.58 MHz was approximately  $91 \cdot \sin(\text{THETA})$  as had been previously predicted by the "DSL" program. This limit could have been determined by simply adding the magnitude of 63 to itself to realize that it would produce an overflow condition of 128. It was thought, however, that the adding and shifting of the filter with the added dividers might produced some higher limit. Indeed, if the frequency was varied slightly from that of the in-phase frequency of 3.58 MHz (e.g. 3.5 MHz), it was found that a slightly higher input magnitude could be used without producing an overflow condition.

This concluded the necessary implementation of the filter at the Fortran level and its accompanying analysis. Without this step in the design process, the implementation at the lower level language of the bit-slice would certainly have been more difficult. Before leaving this section, it should be pointed out that one further step was taken in the analysis of the quantization noise introduced by the filter. The rational polynomial form of the filter was changed to run as fixed point and the output data was compared to the



shift-and-add fixed point output data. Although it appeared from [Ref. 34] that the rational polynomial form of the filter might introduce additional quantization noise, there was no observable difference in the output data. This may be attributable to the fact that the coefficients of the filter are already of integer form.

#### IV. BIT-SLICE IMPLEMENTATION

##### A. INTRODUCTION

The Am29203 evaluation board was used to investigate the effectiveness of implementing the FIR filter in a bit-slice design. The Am29203 evaluation board is a tool whereby a designer may learn and develop the skills needed to design with components of the Am2900 family, keeping in mind that the board would not be used in an actual implementation. AMD offers excellent documentation of the board through its Am29203 Evaluation Board User's Guide which offers many step-by-step examples of using the three major components of the evaluation board. The function and utility of these components were briefly introduced in Chapter II. Once the bit-slice components and the microprogramming of these bit-slice components are fully understood, the user may then develop and analyze microprograms through the use of a monitor using a screen-oriented terminal. The relationship of the 'monitor' to the system is shown in Figure 4.1. The 'monitor' should be treated as a separate system from the primary system and except for the terminal commands, its architecture and details of execution should be transparent to the user. Using the 'monitor' commands, the user is able to load and display main memory, micro memory (control store), and registers and then run a loaded microprogram by

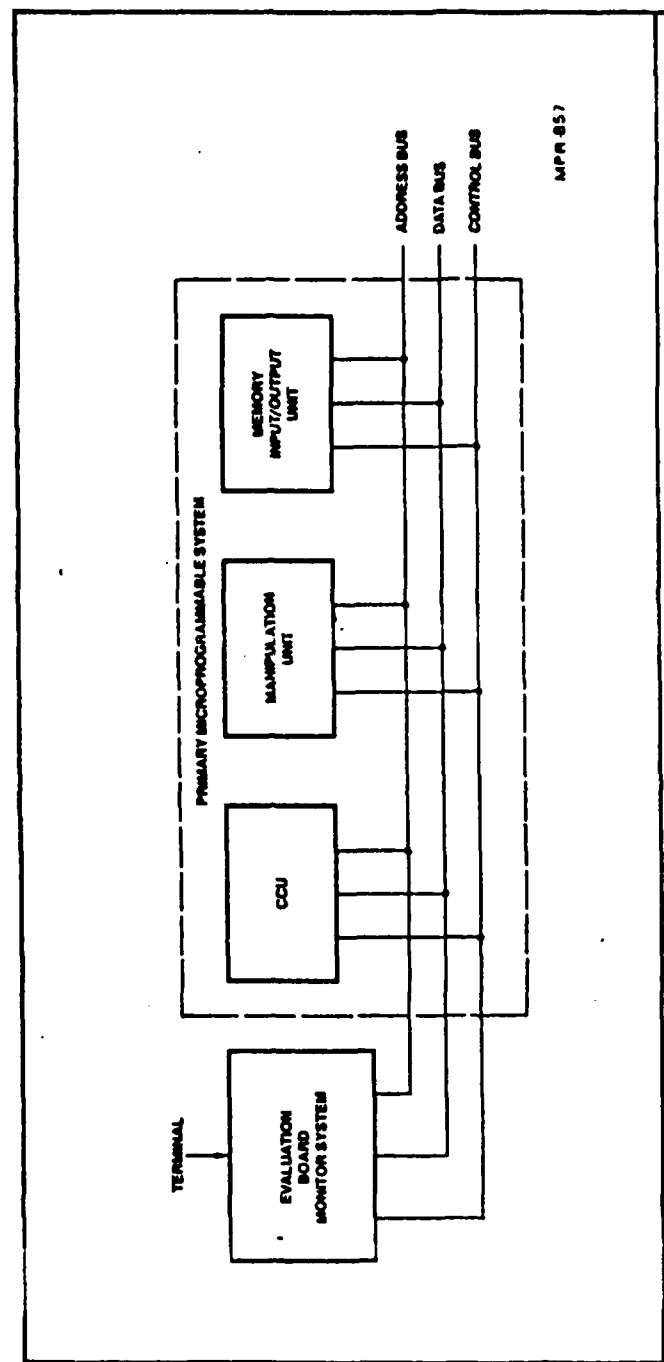


Figure 4.1 Evaluation Board Organization [Ref. 7:p. 3.2]

stepping through it or by using set breakpoints. [Ref. 7:pp. 4.2-4.9]

Previous work done in the area of bit-slice at the Naval Post Graduate School by Morris Bennett Stewart II [Ref. 15] used a dummy terminal for entering and analyzing programs. The disadvantages to this approach are that programs must be entered by hand, greatly reducing the scope of the programs which can be entered, there is no memory capability for retaining programs and there is no method for printing data. The preliminary work which therefore had to be done before implementing the FIR filter in bit-slice was to emulate the dummy terminal using an IBM PC and the commercial software Smartcom II. This proved to be a somewhat difficult task due to the lack of documentation provided and the lack of expertise in this area given by personal conversation with AMD. Once implemented however, the programs could then be created using a personal editor to write ASCII files and stored to disk. Then these files could be downloaded to the Am29203 evaluation board using Smartcom II. This greatly facilitated the ability to run, analyze and change the FIR filter microprogram. The additional feature was the ability to record a working session or print out data stored in micro and macro memory through the use of the printer. A brief explanation and documentation of implementing the monitor through the use of Smartcom II is provided in Appendix A.

With the above hardware and software in place, the FIR filter could now be adequately implemented using the Am29203 evaluation board. This chapter will describe the use of the major components of the evaluation board and then present the macro and microprogram used to control these components and thereby implement the FIR filter using bit-slice methodology. Finally, a time comparison between Fortran implementation and bit-slice implementation will be given.

## B. USE OF EVALUATION BOARD COMPONENTS

Chapter II introduced the basic architecture and operation of the three major components of the evaluation board which are directly controlled by the micro word: the Am2910 12 bit sequencer; the 16 bit ALU consisting of four 4-bit Am29203 CPU slices; and the Am2904 control unit. A good understanding of these components and the micro fields used to control them are required before a designer can write any microprograms using bit-slice. For example, a simple add at the macro level may take several steps in microcode. Although a novice designer may be able to "get the job done", it takes an expert designer to truly optimize and get the full time saving benefits of the microcode. It has been estimated that it may take fully two years or more before a designer will be able to design easily using bit-slice methodology.

The Am2910 field can be taken as an example of how the microword controls the components directly. The basic

concept which must be understood about the Am2910 is that it simply sequences the microinstructions, primarily through the use of loops, counters and stack register. The communication interface with the Am2904 provides the necessary condition code status for the conditional branching. The function of the Am2910 is probably best understood by studying the sixteen Am2910 instructions shown in Figure 4.2. These sixteen instructions are represented by the sixteen hexadecimal values 0-F and in the case of the Am29203 evaluation board, make up the field of the last 4 bits of the evaluation board's 48 bit microword. Although it appears to be a fairly simple concept, [Ref. 7:pp.6.1-6.18] uses an entire chapter to discuss the use of this field and yet barely even touches on the subject of the interrelationships required between the Am29203 and Am2904 fields.

As can be seen from the above discussion, it would be impossible in this format to offer a brief explanation of each of the fields and overlaid fields of the 48 bit microword of the evaluation board. It is sufficient to say here that the Am29203 performs the boolean logic operations for executing the desired arithmetic or logic functions and the Am2904 provides the status testing and shifting of the operations.



## C. BIT-SLICE IMPLEMENTATION OF THE FIR FILTER

### 1. General Goals

The portion of the Fortran program implementation of the FIR filter to be implemented in bit-slice is repeated here in Figure 4.3. The DO loop portion of the code may be somewhat of an artificiality, imposed by subjecting the filter to a limited number of data points for testing purposes, but was necessary in the testing of the bit-slice code as well.

It was felt that the major goal of implementing this filter in bit-slice would be to retain as much data in the sixteen Am29203 general purpose registers as possible. This would greatly reduce the most time-consuming task of reading and writing required data to be manipulated from RAM. The ideal situation of course would be to retain all of the 13 shifting (delay) data points in the general purpose registers of this 13th order filter. This seemed somewhat impossible since generally only 7 of the 16 general purpose registers are actually available for general data manipulation. However, by not using the standard macro instructions provided with the evaluation board, it appeared potentially possible to free up 15 of the 16 general purpose registers for this specific application, with the 16th register needed for the macro program counter. With this in mind, the filter is rewritten to use the registers as variables in the delay equations as shown in bold type in



```

C -----
SUBROUTINE FUNCT (X,Y,N)
  INTEGER X(100),Y(100),Y1,Y2,Y3,Y4,Y5
  INTEGER X1/O/,X2/O/,Y14/O/,Y13/O/,Y12/O/,Y11/O/,Y23/O/
  INTEGER Y22/O/,Y21/O/,Y31/O/,Y41/O/,Y52/O/,Y51/O/
  INTEGER K,N
  DO 50 K=1,N
    Y1=X(K)-X2
    Y1=Y1/2
    X2=X1
    X1=X(K)
    Y2=Y1+Y14
    Y14=Y13
    Y13=Y12
    Y12=Y11
    Y11=Y1
    Y3=Y2+Y23
    Y23=Y22
    Y22=Y21
    Y21=Y2
    Y4=Y3-Y31
    Y31=Y3
    Y5=Y4-Y41
    Y41=Y4
    Y(K)=Y5-Y52
    Y52=Y51
    Y51=Y5
  50 CONTINUE
  RETURN
  END
C -----

```

Figure 4.3 Fortran Routine of FIR Filter

Figure 4.4. This would then leave two registers available for incoming and outgoing data from RAM. Now looking at the implementation of the new version of the filter, it is seen that 6 variables are still required for each of the six stages of the filter and one is needed for the input to the filter. However, it was discovered that by alternating the names of the variables, the number of variables required could be reduced from 7 down to 2 as shown in bold type in Figure 4.5. This meant that an input to the filter could be read from RAM, then entirely manipulated in the general purpose registers through the six stages of the filter, with the output of the filter then written to RAM. These changes reduced the total number of reads/writes required to RAM from 38 down to only 2. The specific time savings will be presented later in this chapter.

## 2. Bit-Slice Macro and Micro Instructions

In the design of the FIR filter in bit-slice, the macro instruction and micro instructions were developed somewhat simultaneously and it is difficult to separate the two. However, logically the macro instruction for the filter should be presented first. A typical instruction sequence, using 30 points of input data to the filter for testing of the code, is shown in Figure 4.6. This instruction sequence is shown as printed out by the macro memory display function of the evaluation board 'monitor'. It displays the data in the form of eight memory locations

$Y1-X-R2$   
 $R2-R1$   
 $R1-X$   
 $Y2-Y1+R6$   
 $R6-R5$   
 $R5-R4$   
 $R4-R3$   
 $R3-Y1$   
 $Y3-Y2+R9$   
 $R9-R8$   
 $R8-R7$   
 $R7-Y2$   
 $Y4-Y3-RA$   
 $RA-Y3$   
 $Y5-Y4-RB$   
 $RB-Y4$   
 $Y-Y5-RE$   
 $RE-RD$   
 $RD-Y5$

Figure 4.4 FIR Filter Using 13 Registers of the Am29203

$RO-RC-R2$   
 $R2-R1$   
 $R1-RC$   
 $RC-RO+R6$   
 $R6-R5$   
 $R5-R4$   
 $R4-R3$   
 $R3-RO$   
 $RO-RC+R9$   
 $R9-R8$   
 $R8-R7$   
 $R7-RC$   
 $RC-RO-RA$   
 $RA-RO$   
 $RO-RC-RB$   
 $RB-RC$   
 $RC-RO-RE$   
 $RE-RD$   
 $RD-RO$

Figure 4.5 FIR Filter Using 15 Registers of the Am29203

>DM ADDR:0200

0200	-	01F0	001E	0000	3E00	FF00	C200	0300	3E00
0208	-	FB00	C200	0600	3E00	F800	C200	0900	3E00
0210	-	F500	C300	0D00	3D00	F200	C300	1000	3C00
0218	-	EE00	C400	1300	3B00	EB00	C600	1600	3A00

Figure 4.6 Macrocode for FIRFILT Routine

per line. The first instruction, 01F0, is the actual "calling" instruction of the filter and as an example, might be given the macro instruction mnemonic of FIRFILT which would represent the call for the FIR filter microprogram. The first two digits, 01, represent the opcode portion of the instruction and through the mapping PROM, maps to the micro-address 0004 where the microprogram sequence of instructions for the filter is located. The second two digits, F0, represent the source and destination registers, register F and register 0 respectively, to be used later in the instruction register in the micro code. The next macro instruction in the sequence is actually not an instruction but is the hexadecimal representation of the number of data points which follow. This number will be fetched to the Q register at the microprogram level and will be used as a counter for the number of data points to process in the filter. The next 30 instructions are then the 30 data points represented in hexadecimal form which will be processed by the filter at the micro instruction level.

Now to run this macro instruction, FIRFILT, at the micro instruction level, the macro instruction must first be fetched from macro memory into the microprogram instruction register (IR). The standard instruction fetch [Ref. 7:p. 9.7] is used here and the three micro instructions needed are shown in Figure 4.7. A detailed explanation of all micro instructions is provided in Appendix B. Basically, the first instruction loads the instruction to the IR, the second instruction updates the PC (macro program counter located in register F), and the third instruction maps the instruction to the microroutine which will execute the instruction. Two notes are made here. First, this standard instruction fetch also places a copy of the instruction in register D which may be needed in some microprograms. In this case however, it will be overwritten by the filter microroutine. Secondly, the Am29203 chip allows the fetching of an instruction and the updating of the PC to occur simultaneously, however the architecture of the evaluation board does not [Ref. 7:p. 9.8].

IFETCH:	0200	-	084F	3FD6	FFDE
	0201	-	0044	7FFF	FFFE
	0202	-	FFFF	FFFF	FFF2

Figure 4.7 Microroutine IFETCH

As stated earlier, the instruction is mapped to the micro address location 0004 and this microroutine is shown in Figure 4.8. Although these 49 lines are one microroutine, it is broken down into several sections and labeled with mnemonics to break this long routine into easily described sections and to create an easy method for locating a particular section of the microroutine in Appendix B. This microroutine is described in its mnemonic labeled sections as follows:

**LDCNTR**--Loads the Q register with the counter as taken from macro memory. The PC is not updated here but put in the loop.

**CLREG**--Clears 13 registers for the implementation of the 13 delay equations.

**LOOPBEG**--Marks the beginning of the loop. It updates the PC and brings in the first data point.

**STAGE1**--This marks the actual beginning of the filter in microcode. Address H#014 provides the first adder stage, H#015 and H#016 provide the call for the microroutine to divide the result by 2 (shifter microroutine) depending on whether the result is positive or negative, and H#018-019 provide the microcode for the 2 delay equations following the adder. (Note: H stands for hexadecimal)

**STAGE2**--Provides the second stage of the filter with adder, call for divide by 2 microroutine, and 4 delays.

**STAGE3**--Provides the third stage of the filter with adder, call for divide by 2 microroutine, and 3 delays.

**STAGE4**--Provides the fourth stage of the filter with adder, call for divide by 2 microroutine, and 1 delay.

**STAGE5**--Provides the fifth stage of the filter with adder and one delay.

```

LDNTR: 0004 - 084F FF03 FFCE
        0005 - 0064 3FFF FFCE
CLREG: 0006 - 0248 FFFF FF1E
        0007 - 0248 FFFF FF2E
        0008 - 0248 FFFF FF3E
        0009 - 0248 FFFF FF4E
        000A - 0248 FFFF FF5E
        000B - 0248 FFFF FF6E
        000C - 0248 FFFF FF7E
        000D - 0248 FFFF FF8E
        000E - 0248 FFFF FF9E
        000F - 0248 FFFF FFAE
        0010 - 0248 FFFF FFBE
        0011 - 0248 FFFF FFDE
        0012 - 0248 FFFF FFEE
LOOPSES: 0013 - 0044 7FFF FFFE
        0014 - 084F FF03 FFCE
STAGE1: 0015 - 8041 507F F2CE
        0016 - FFFF FFFF E50C
        0017 - FFFF DF09 E705
        0018 - 0246 3FFF F12E
        0019 - 0246 3FFF FC1E
STAGE2: 001A - 4043 107F F6CE
        001B - FFFF FFFF E10C
        001C - FFFF DF09 E305
        001D - 0246 3FFF F56E
        001E - 0246 3FFF F45E
        001F - 0246 3FFF F34E
        0020 - 0246 3FFF F03E
STAGE3: 0021 - 8043 107F F9CE
        0022 - FFFF FFFF E50C
        0023 - FFFF DF09 E705
        0024 - 0246 3FFF F89E
        0025 - 0246 3FFF F78E
        0026 - 0246 3FFF FC7E
STAGE4: 0027 - 4041 507F FACE
        0028 - FFFF FFFF E10C
        0029 - FFFF DF09 E305
        002A - 0246 3FFF FOAE
STAGE5: 002B - 8041 507F FBCE
        002C - 0246 3FFF FC8E
STAGE6: 002D - 4041 507F FECE
        002E - 10C4 3FD4 FFCE
        002F - 0246 3FFF FDEE
        0030 - 0246 3FFF F0DE
DECCNTR: 0031 - 8244 3FFF FFFE
        0032 - 0030 7FFF FFOE
        0033 - 0064 107F FFOE
        0034 - FFFF D4D9 C133

```

Figure 4.8 Microcode for FIR Digital Filter

**STAGE6**--Provides the sixth stage of the filter with adder and 2 delays. This marks the end of the filter in the microroutine. Address H#02E places the filter output data back into the macro memory location pointed to by the PC which in doing so, writes over the previous input data given at the beginning of the loop.

**DECCNTR**--Decrements the counter and loops back to address H#013 if the counter is not zero.

The microroutine above called another microroutine for the dividing by 2, which is actually accomplished by shifting in the two's complement implementation, without offering an explanation. It would seem that a shift to divide a positive or negative number by 2 using two's complement arithmetic should require only one instruction in microcode to implement the proper shift. Indeed, this would normally be the case if the sign of the number being shifted is known ahead of time. In fact, as shown in the example of Chapter II, it is possible to accomplish a shift and add in a single instruction. Two problems arise in this particular implementation however. First, it is not known ahead of time whether or not the operands will be positive or negative. It should also be kept in mind that some of the adder stages are actually subtractors. The second and biggest problem in this case however comes from the restriction imposed of using only 8 of the 16 bits available in the ALU. A clearer understanding of two's complement arithmetic would have saved a great deal of time in this area. For example, using decimal integer arithmetic, a three divided by two would result in a one with the 0.5



being truncated. This is also true of two's complement arithmetic where the divide by 2 is accomplished by shifting all the bits to the right by one and with a zero fill at the most significant bit. However, if the eight bits of data were placed in the upper eight bits as was first done, a division by two in this case could cause a one to be transferred into the upper bit of the lower eight bits. For example, dividing the hexadecimal #0300 by two would result in H#0180, which is indeed the correct result but it is not the desired result of H#0100. To transfer the unwanted one out of the upper bit of the lower eight bits requires eight shifts to the right with zero fill to the left and then eight shifts back to the left. This would then produce the desired result of #0100. The case of dividing a negative number in two's complement arithmetic is a bit more complicated. For example, using integer decimal arithmetic, dividing the number -3 by 2 would produce a result of -1. In two's complement arithmetic however, where the divide by two is accomplished by shifting to the right with a one being filled in the most significant bit, the result would be a -2. To account for this difference, a one must be added to the operand, before the shifting, to produce a correct result of -1. Therefore, to accomplish the correct result using only the upper eight bits for entering data, the operand must first be shifted to the lower eight bits with ones being filled in the upper eight bits. Then a one

is added to the operand and the final shift to the right with one fill in the most significant bit is accomplished. This places the correct result of dividing the operand by two in the lower eight bits. The result is then shifted to the left eight bits to place the result in the upper eight bits.

A much more straightforward approach is obtained by placing the incoming data in the lower eight bits. The problem here, however, is that in the case of incoming negative data, the upper eight bits must be filled with ones to make the number in the lower eight bits negative. Again, the rules for dividing a negative number by two in this case still apply.

In this particular application, the first approach presented of placing the data in the upper eight bits was used, primarily for two reasons. First, the number of operations used here was not of importance since this was an artificiality which was placed on the problem using the hardware which was available. With this in mind, the first solution allowed for the solution of a much more interesting problem and allowed for a broader knowledge of the bit-slice to be obtained. Secondly, this approach originally offered an easier method for entering the data. This was later shown not to be valid for entering large amounts of data through the aid of the computers. A file in Fortran in hexadecimal form can be created using 'z' in the FORMAT

statement when writing to a file. This hexadecimal file will be in the correct form which can then be downloaded to a disk. Once on the disk, the file can be transferred to the bit-slice RAM using Smartcom II.

The set of microprogram routines for accomplishing the divide by two in the upper eight bits for both the positive and negative cases is shown in Figure 4.9. This set is for an operand which is in the RC register. Another set identical to this, only with '0' specified, was used when the operand was stored in the R0 register. A straight forward approach was used and the code was not optimized for time, as was done in the filter microroutine. A detailed explanation of the micro instructions is included in Appendix B.

#### D. FORTRAN AND BIT-SLICE IMPLEMENTATION SPEED COMPARISONS

As pointed out in several of the references, including the evaluation board user's guide [Ref. 7], the objective of a full timing analysis is to find the longest path and then use that time to determine the minimum clock period for the given design. With this in hand, there are several alternatives to the design. If the time used is acceptable, one alternative would be to leave the clock period as it is. If it is not acceptable, there are many alternatives to improve the overall time used. One method would be to look for ways to improve the algorithm or code used. Another

```

1044 107F FFCE
FFFF 0509 E233
0004 3FE0 FFCE
0004 3FE0 FFCE
0004 3FE0 FFCE
0004 3FE0 FFCE
0004 3FE0 FFCE
0004 3FE0 FFCE
0004 3FE0 FFCE
0004 3FE0 FFCE
0004 3FE0 FFCE
0004 3FE0 FFCE
0004 3FE0 FFCE
0084 3FE0 FCCE
0084 3FE0 FCCE
0084 3FE0 FCCE
0084 3FE0 FCCE
0084 3FE0 FCCE
0084 3FE0 FCCE
0084 3FE0 FCCE
0084 3FE0 FCCE
0084 3FE0 FCCE
FFFF E4F9 FFFA

```

POSITIVE CASE

```

0004 3FE1 FFCE
0004 3FE1 FFCE
0004 3FE1 FFCE
0004 3FE1 FFCE
0004 3FE1 FFCE
0004 3FE1 FFCE
0004 3FE1 FFCE
0004 3FE1 FFCE
0004 7FE1 FFCE
1044 107F FFCE
FFFF 0509 E43E
0084 3FE0 FCCE
0084 3FE0 FCCE
0084 3FE0 FCCE
0084 3FE0 FCCE
0084 3FE0 FCCE
0084 3FE0 FCCE
0084 3FE0 FCCE
0084 3FE0 FCCE
0084 3FE0 FCCE
FFFF E4F9 FFFA

```

NEGATIVE CASE

Figure 4.9 Shifting Microroutines

would be to use faster components where needed such as using faster memories. One faster component which might be used is a variable clock circuit. It is used to lengthen or shorten the clock period depending on the length of the timing path for each instruction. [Ref. 7:p. 6.13]

The primary method used in this study to improve the overall time was that of seeking ways to improve the algorithm and code used. Other methods are also considered in this section and the data obtained is shown in Table II. The first comparison obtained is that between the Fortran implementation on the VAX and that of the improved 16 register microcode implementation using the fixed and extremely slow clock period of the evaluation board. Improved microcode, here and in Table II, refers to the microcode which was designed for this special FIR filter application which takes full advantage of the 16 registers of the Am29203. The timing of the Fortran was obtained by using the subroutine "jcput". The code for this subroutine and its placement in the Fortran program can be found in Appendix C. The VAX routines LIB\$INIT\_TIMER and LIB\$SHOW\_TIMER [Ref. 13] can also be used to obtain estimates of the time required and is given in increments of 10 milliseconds. The time obtained for 100 iterations of the filter was found to be 10 milliseconds or 100 microseconds per iteration. Using even the slowest form of the bit-slice, using the fixed clock period of the

TABLE II  
TIME COMPARISONS FOR FIR IMPLEMENTATIONS

<u>Method</u>	<u>Time (microseconds)</u>
Fortran Implementation	100
Evaluation Board Provided	20
Bit-Slice Code	49 inst. * 408ns
Improved Bit-Slice Code using	11
29203 evaluation board	27 inst. * 408ns
Evaluation Board Provided	14.85
Bit-Slice Code with PROM	27 inst. * 408ns 25 inst. * 153ns
Improved Bit-Slice Code using	4.64
29203 evaluation board with PROM	2 inst. * 408ns 25 inst. * 153ns
High speed Am2900 family	2.65
Bit-Slice	
VITESSE's Gallium Arsenide	.78
Bit-Slice	

evaluation board of 408 nanoseconds, the time was found to be only 11 microseconds per iteration of the filter, almost 10 times faster than the Fortran implementation. This was obtained simply by multiplying the 27 instructions of microcode of the filter, including the instructions for updating the PC and counter, by the 408 nanosecond clock period.

Next, a comparison was made between that of the improved microcode and that of the provided code of the Am29203 evaluation board. It is somewhat difficult to determine the exact number of instructions needed using the provided code without actually writing and testing the routine, however it is estimated that it would take approximately 49 instructions for a total time of 20 microseconds using this approach. The improved code therefore has an approximate time savings of 45% over the provided code.

One of the goals when improving the microcode was to minimize the number of instructions which required a read from RAM, such as those required when inputting data. In his study of bit-slice, Morris Stewart [Ref. 15] documents how the fixed instructions of the microcode could be placed in a faster PROM to shorten the time path of these instructions. Then a variable clock generator could be used to shorten the clock period of these instructions to 153 nanoseconds. The improved microcode can now take advantage of this since only

2 of the 27 instructions require an access to RAM. The total time now required is 4.64 microseconds as shown in Table II. For the provided code, if it is assumed that the 13 delay variable addresses are in microcode or PROM, this routine would still require 27 of the 49 instructions to address the RAM for a total time of 14.85 microseconds. The improved microcode clearly has an advantage in this case and results in a time savings of nearly 70% over the provided code.

Finally, a look is taken at how new bit-slice devices presently on the market could be used to improve the overall time of the filter implementation. Figures 4.10 and 4.11 provide control loop and data loop comparisons of AMD's high speed versions of the Am2900 family to VITESSE's Gallium Arsenide 2900 family devices. As can be seen from these figures, the high speed devices require a minimum cycle time of 98 nanoseconds while the Gallium Arsenide devices require a minimum cycle time of 29 nanoseconds. Now, using these speeds, one iteration of the bit-slice filter will require 2.65 microseconds and 0.78 microseconds respectively. This is over 100 times faster than the Fortran implementation and would result in a significant amount of time savings with the large amount of data iterations that would be used in an actual filter implementation. It should be noted that these last comparisons are made using only a single level pipeline,



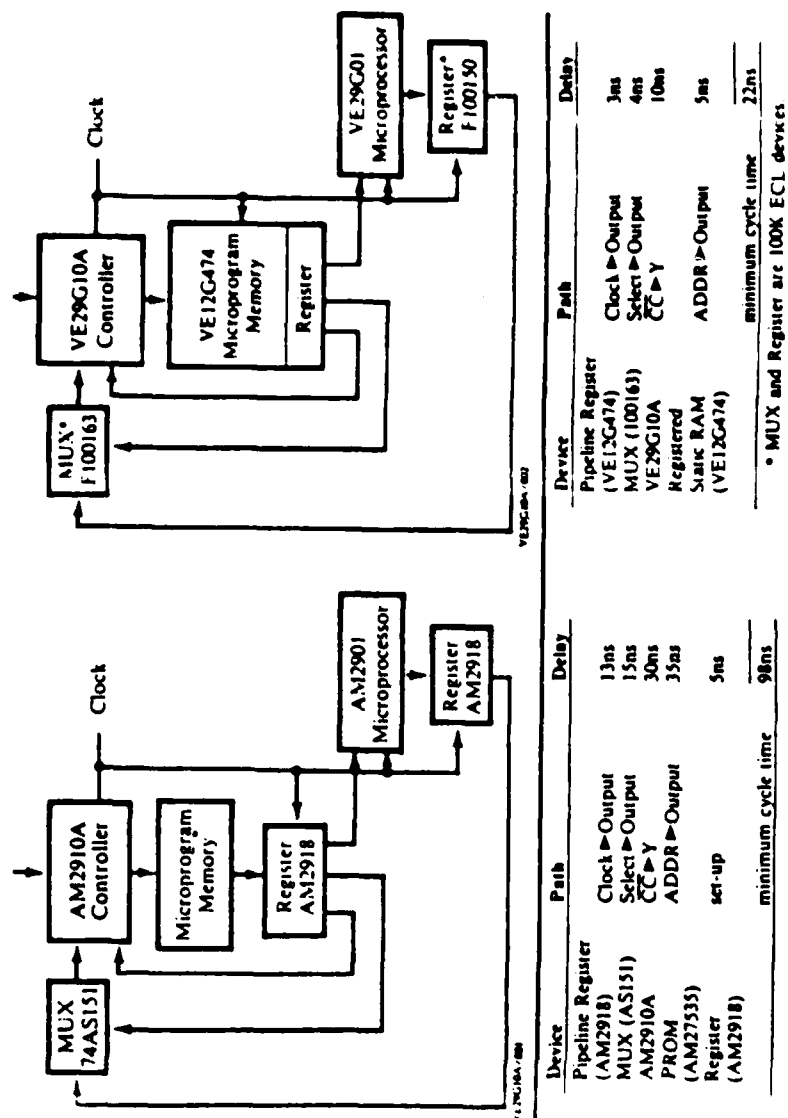


Figure 4.10 Comparison of Minimum Control Loop Cycle Time of 16 Bit System, Single Level Pipeline [Ref. 16]

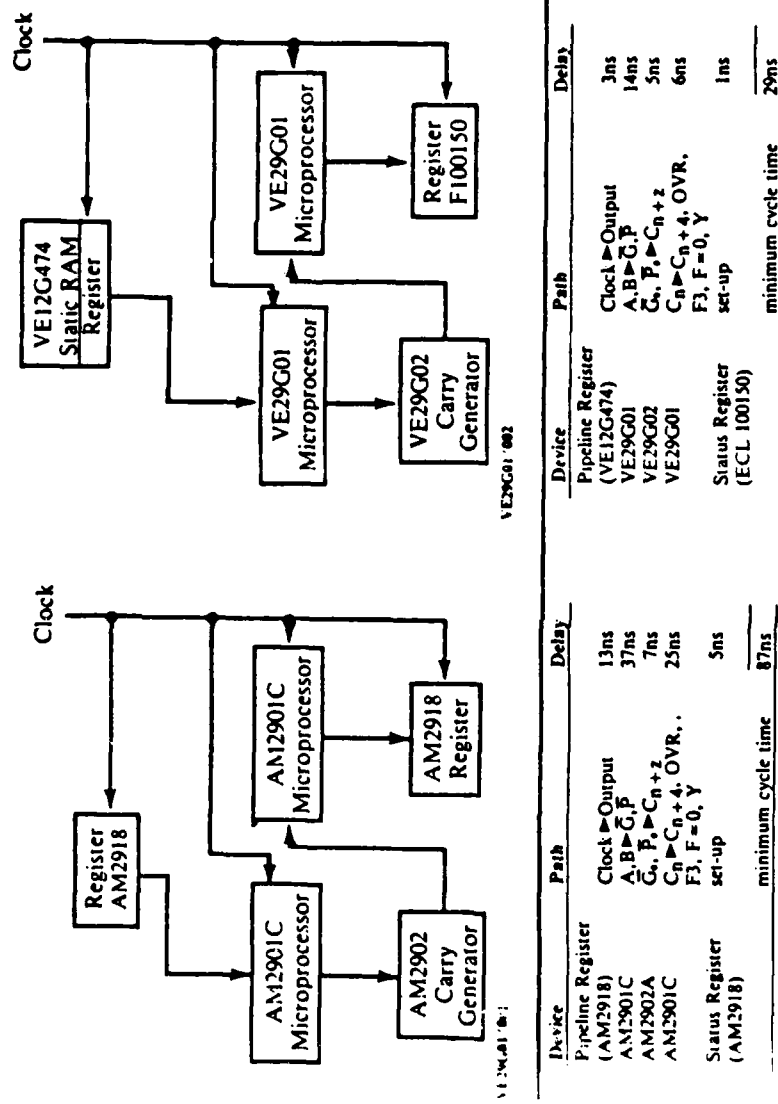


Figure 4.11 Comparison of Minimum Data Loop Cycle Time of 16 Bit System, Single Level Pipeline [Ref. 16]

AD-A184 895

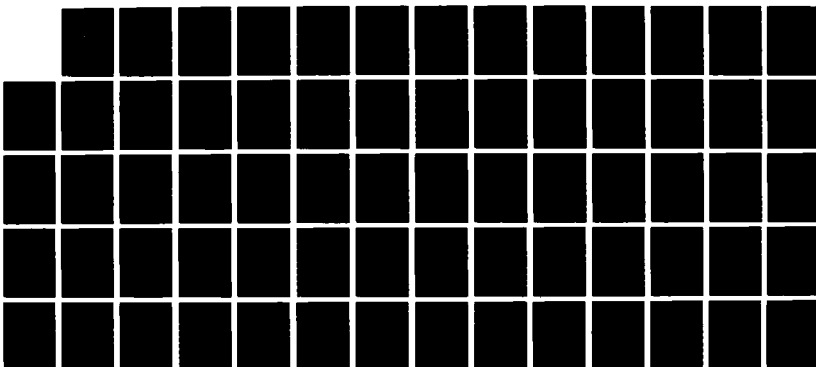
IMPLEMENTATION OF AN FIR BAND PASS FILTER USING A  
BIT-SLICE PROCESSOR(U) NAVAL POSTGRADUATE SCHOOL  
MONTEREY CA D H PURDY JUN 87

2/2

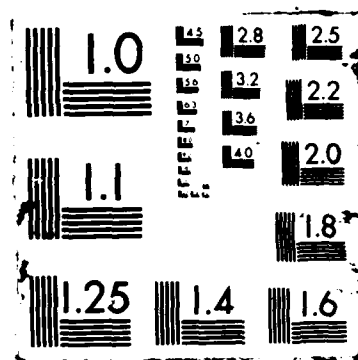
UNCLASSIFIED

F/G 12/6

NL



IND  
H. I  
J. P



whereas in earlier comparisons, a three level pipeline is used as delineated by White [Ref. 5].

## V. CONCLUSIONS

One of the strongest arguments against the use of bit-slice designs is the time in which it takes to design with these devices as compared to other methods. The proposed trade-off with the longer design time is the ability to achieve greater speeds thereby producing dividends in processing large volumes of data over long periods of time. In this limited study, however, it appeared that most of the time in designing this simple FIR filter was spent toward gaining a working knowledge of the bit-slice components and overcoming the needed skills in working with two's complement arithmetic. It seemed that once this working knowledge is obtained, an expert designer should be able to easily design such a simple circuit in a small amount of time. The complexity of the bit-slice language necessarily prohibits its use as a general design tool but its benefits in speed have a range of application when left to the expert designer. As seen from Chapters II and IV, the bit-slice devices easily approach super-computer speeds and yet at a small fraction of the cost. It should be pointed out here that only a limited working knowledge was gained during this study and there are certainly many more aspects and benefits which could be learned through further study. The microcode

implementation presented for the FIR filter probably is not optimized and could be improved upon.

One of the problems of bit-slice methodology is its use in limited studies such as this. For example, for a follow on study in this area, a researcher would have to go through the same difficult process of learning and obtaining a working knowledge of the bit-slice language before any further work could be done. This obviously limits the scope of the study and impedes the progress of research which can be made. The bit-slice evaluation board and accompanying user's guide is an invaluable tool in learning the application of the bit-slice components and it is difficult to imagine how this material might be presented in a more condensed form in order to achieve a faster learning process.

As with any research, an analysis must eventually be made as to what conclusions can be made and what questions were raised during the research which remain unanswered. In this study, a thirteenth-order FIR filter was successfully implemented in bit-slice using only shifters and adders. The two major goals of implementing the filter on the evaluation board and using a computer to download files to the evaluation board were achieved. The time savings using the bit-slice implementation far out-weighed the time spent in designing it. It was also seen that the implementation could be limited to eight bits of accuracy without

significantly affecting the results. One question which came up during this implementation which could have been further researched was how the limitation to 8 bits of accuracy on the implementation truly affected the noise, especially with the introduction of the 5 stages of shifters or dividers. The main questions which were raised during research and remain to be answered however, were: why were the six stages of the filter put in the order in which they were in; how does this order affect the quantization effects; and how was this thirteenth order filter reduced to a filter using six stages with coefficients of one?

In summary, the bit-slice methodology provides extremely useful devices for achieving increased speeds in specific applications, especially in those applications of high speed graphics where large amounts of data to be processed benefit from the improved processing times. Because of its versatility in implementing any given instruction set, it should not be ruled out as a design tool based merely on the time required to design with it.



## APPENDIX A

### TERMINAL EMULATION USING SMARTCOM II

The commercial software SMARTCOM II by Hayes [Ref. 17] was used on the IBM PC to emulate the user terminal for the monitor system of the Am29203 evaluation board. This appendix will only document the problems encountered in using SMARTCOM II and the necessary configurations which must be made to use SMARTCOM II to communicate with the evaluation board using the IBM PC. It should be noted here that only the SMARTCOM II software was needed for the configuration and the SMARTCOM II modem was neither used nor installed.

The primary problem encountered in using SMARTCOM II was not the configuring of the software, although this did prove to be somewhat time consuming. The major problem was the interconnection of the hardware. From the advice of technicians consulted and two references used, including the SMARTCOM II manual, it appeared that a null modem would have to be used between the IBM PC and the evaluation board since both are computers and have DCE connection ports. In fact, a null modem was constructed with pins 2 and 3 crossed to ensure that both computers could send and receive properly. The problem discovered however, was that SMARTCOM II was changing the signal internally since the DCE port of the IBM

PC was behaving as if it were a DTE port. With this discovery made the only connection between the two computers required was a straight line gender changer.

Once SMARTCOM II has been entered, there are several screens which can be entered to change the required parameters. First, the Batch Set Directory, a listing of all batch sets (communication devices available), must be entered to list the evaluation board as one of the options available. This is shown in Figure A.1. Next, the Configuration Screen must be changed to reflect the equipment being used as shown in Figure A.2. Finally, the Parameter Screen lists the variables or parameters for each particular communication environment. Figure A.3 shows the parameter screen for the Bit-slice evaluation board environment. These changes do not have to be made for every entry into the SMARTCOM II software program.

The Menu Screen shown in Figure A.4 is used to communicate back and forth between the bit-slice evaluation board environment and the SMARTCOM II environment. Option 1 is selected to enter the On-Line Screen and in this mode, the IBM PC monitor and keyboard appear to the evaluation board and user as if it were an ordinary terminal. To terminate the session or bring in a data or program file stored on disk, F1 is pressed to return to the SMARTCOM II menu screen for the appropriate selection.

# Smartcom II

Hayes Microcomputer Products, Inc.

1. Begin Communication
2. Edit Set
3. Select File Command
- A,B - Change Drive
- \*. Receive File
- \*. Send File
6. Change Configuration
7. Change Printer Status (OFF)
- \*. Select Remote Access (OFF)
9. Display Disk Directory (OFF)
0. End Communication/Program

Press F2 For Help

Enter Selection: 1  
Enter Label: P

## Communication Directory:

A - CompuServe Direct	J - OAG EE Telenet	S - CompuServe Datapac
B - CompuServe Telenet	K - OAG EE Tymnet	T - DJN/R Datapac
C - CompuServe Tymnet	L - OAG EE UNINET	U - KNOWLEDGE INDEX Data
D - DJN/R Telenet	M - THE SOURCE Direct	V - OAG EE Datapac
E - DJN/R Tymnet	N - THE SOURCE Telenet	W - THE SOURCE Datapac
F - DJN/R UNINET	O - THE SOURCE UNINET	X - Test Set
G - KNOWLEDGE INDEX Tel	P - bit_slice_board	Y - Remote Access
H - KNOWLEDGE INDEX Tym	Q - AERO.NET	Z - Standard Values
I - MCI Mail	R -	

9:30 pm

Sunday May 31, 1987

Figure A.1 Communication Directory

```

PRINTER :                                     CONFIGURATION      Press F2 For Help
Port: PARALLEL
Serial Protocol:
Baud:
Remove Extra Line Feeds: YES
Add NULs: 0
SMARTMODEM :
Port: NONE
Dialing Method: PULSE
Pause Time For Comma: 2 ( 0-255 seconds )
Touch-Tone Timing: 70 ( 50-255 0.001 seconds )
Wait For Dial Tone: 2 ( 2-255 seconds )
Wait For Carrier Signal: 30 ( 1-255 seconds )
Recognize Carrier Signal: 6 ( 1-255 0.1 seconds )
Carrier Loss To Hangup Time: 7 ( 1-254 0.1 seconds )
Speaker Status: ON UNTIL CARRIER
Telephone Jack Type: RJ11
SPECIAL VALUES : Default Set: P
Available Disk Drives: AB
Monitor and Adapter: COLOR/GRAPH. DISP. ADAPTER/COLOR MONITOR
Log-on Message: Smartcom II - IBM Personal Computer
Direct-Connect Port: COM2:

```

9:37 pm

Sunday May 31, 1987

Figure A.2 Configuration Screen

Name of Set: P - bit\_slice\_board      **PARAMETERS**

Press F2 For Help

```

TRANSMISSION PARAMETERS
Duplex: FULL
Connection Type: DIRECT 2400
Character Processing: FORMATTED
Show Control Codes: NO
Page Pause: NO
Show Status Lines: YES
Confidential: NO
Include Line Feeds: NO
Character Delay: 30 (0.001 sec.)
Line Delay: 30 (0.01 sec.)
Character Format: 8 DATA + NONE + 1 STOP
Emulator: TELEVIDEO SUBSET

TELEPHONE PARAMETERS
Answer On Ring: 0
Remote Access: NONE Password:
Phone Number:

KEYBOARD DEFINITIONS
Escape Key: 128 (F1)
Help Key: 129 (F2)
Printer Key: 130 (F3)
Capture Key: 131 (F4)
Macro Prefix Key: 132 (F5)
Break Key: 133 (F6)
Break Length: 35 (0.01 sec.)
Protect Key: 134 (F7)

PROTOCOL PARAMETERS
Receive Time-out: 60 (sec.)
Send Time-out: 10 (sec.)
Error-Free Protocol: HAYES
Stop/Start- Stop Char: 0 (off)
Start Char: 0 (off)
Send Lines- EOL Char: 0 (off)
Prompt Char: 0 (off)

```

9:36 pm

Sunday May 31, 1987

Figure A.3 Parameter Screen

Smartcom II

Hayes Microcomputer Products, Inc.

- |                        |                         |                                 |
|------------------------|-------------------------|---------------------------------|
| 1. Begin Communication | *. Receive File         | 7. Change Printer Status (OFF)  |
| 2. Edit Set            | *. Send File            | *. Select Remote Access (OFF)   |
| 3. Select File Command | 6. Change Configuration | 9. Display Disk Directory (OFF) |
| A,B - Change Drive     |                         | 0. End Communication/Program    |

Press F2 For Help

Press F1 To Return On-Line

Enter Selection: 1

Dials or answers phone with Smartmodem

9:29 pm

Sunday May 31, 1987

Figure A.4 Menu Screen

Since the Smartcom program could only communicate up to a baud rate of 2400, the baud rate of the evaluation board had to be changed from its standard of 4800 baud rate to that of 2400. This is done by simply changing a jumper connection on the evaluation board from W4 to W3.

The use of SMARTCOM II proved to be satisfactory for this study. The biggest inconvenience was that SMARTCOM II had to be completely exited to create a file or edit an exiting file. This proved to be very time consuming when trouble shooting the microroutine. There are now newer software programs on the market which solve this problem and allow the editing of existing files without exiting the emulator. The editor which was used to create the ASCII files was PERSONAL EDITOR 2 by IBM.

## APPENDIX B

### DOCUMENTATION FOR MICROROUTINES

Micro Routine: IFETCH  
Micro address: 0200

BITS	VALUE	EXPLANATION
-----		
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#1	Disable Y output
42-40	Q#0	Operand Sources from RAM
39-36	H#4	Destination to RAM with parity
35-32	H#X	Don't care
31-30	B#00	No carry in
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#01	Select command overlay
19-16	H#6	Instruction fetch
15	B#1	Don't set breakpoint
14	X	Spare/Don't care
13-12	B#XX	Don't care
11-8	H#F	Ra=RF
7-4	H#D	Rb=RD
3-0	H#E	Continue

Resulting Microword: 084F 3FD6 FFDE

Purpose: Fetches instruction from macro memory to instruction register (IR) and register D.

Comments: The Am2904 command, bits 19-16, specify the instruction to be read from macro memory and loaded into the instruction register (IR). The macro memory location is designated by bits 11-8, which is RF, the program counter (PC). A copy of the instruction is also loaded into register D as indicated by bits 7-4. The Am2910 is instructed to continue to the next sequential address.



Micro Routine: IFETCH  
Micro address: 0201

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#0	Operand Sources from RAM
39-36	H#4	Destination to RAM
35-32	H#4	F=S plus carry in
31-30	B#01	Carry in equal to one
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare/Don't care
13-12	B#XX	Don't care
11-8	H#X	Don't care
7-4	H#F	Rb=RF
3-0	H#E	Continue

Resulting Microword: 0044 7FFF FFFE

Purpose: Update PC (increment by one)

Comments: The function specified by bits 35-32 is F=S+ carry in with carry in equal to one. S is specified by bits 7-4 to be RF, the PC. The destination is RF and therefore the PC is incremented by one. The Am2910 is instructed to continue to the next sequential micro instruction.

Micro Routine: IFETCH  
Micro address: 0202

BITS	VALUE	EXPLANATION
47-45	Q#X	Don't care
44	B#X	Don't care
43	B#X	Don't care
42-40	Q#X	Don't care
39-36	H#X	Don't care
35-32	H#X	Don't care
31-30	B#X	Don't care
29-24	Q#X	Don't care
23	B#X	Don't care
22	B#X	Don't care
21-20	B#X	Don't care
19-16	H#X	Don't care
15	B#1	Don't set break point
14	X	Spare/don't care
13-12	B#X	Don't care
11-8	H#X	Don't care
7-4	H#X	Don't care
3-0	H#E	Jump to location mapped by opcode

Resulting Microword: FFFF FFFF FFF2

Purpose: Jump to filter microroutine "BITPRO"

Comments: The Am2910 instruction maps the opcode stored in the IR to the appropriate microroutine location.

Mnemonic: LDCNTR

Micro Routine: Bitpro  
Address: 0004

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#1	Disable Y output
42-40	Q#0	Operand Sources from RAM
39-36	H#4	Destination to RAM with parity
35-32	H#X	Don't care
31-30	B#XX	Don't care
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#01	Select command overlay
19-16	H#3	Read from memory
15	B#1	Don't set breakpoint
14	X	Spare/Don't care
13-12	B#XX	Don't care
11-8	H#F	Ra=RF
7-4	H#C	Rb=RC
3-0	H#E	Continue

Resulting Microword: 084F FFD3 FFCE

Purpose: Load counter from macro memory into register C

Comments: Bits 47-45 specify that the sources Ra and Rb are to be specified by the pipeline at bits 11-8 and 7-4 respectively. Since bits 19-16 specify the command to read from memory, then Ra=RF specifies a macro address and since RF is the program counter, the address specified is the next address in the program. With Rb=RC at bits 7-4, the destination for the value of the macro address is register C. The Am2910 is instructed to continue to the next sequential instruction.

Micro Routine: Bitpro  
Micro address: 0005

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#0	Operand Sources from RAM
39-36	H#6	Destination to Q register with parity
35-32	H#4	F=S plus carry in
31-30	B#00	No carry in
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare/Don't care
13-12	B#XX	Don't care
11-8	H#X	Don't care
7-4	H#C	Rb=RC
3-0	H#E	Continue

Resulting Microword: 0064 3FFF FFCE

Purpose: Load Q register with counter from register C

Comments: Bits 47-45 specify that the sources Ra and Rb are to be specified by the pipeline at bits 11-8 and 7-4 respectively. Since bits 35-30 specify that the function is to be equal to the S operand with no carry in, the value in register C is moved to the Q register as specified by bits 39-36. The Am2910 is instructed to continue to the next sequential micro instruction.

Mnemonic: CLREG

Micro Routine: Bitpro  
Micro address: 0006-0012

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#2	Operand from RAM
39-36	H#4	Destination to RAM with parity
35-32	H#8	F=zero
31-30	B#X	Don't care
29-24	Q#X	Don't care
23	B#X	Don't care
22	B#X	Don't care
21-20	B#X	Don't care
19-16	H#X	Don't care
15	B#1	Don't set break point
14	X	Spare/don't care
13-12	B#X	Don't care
11-8	H#X	Don't care
7-4	H#__	Specify register to be cleared
3-0	H#E	Continue

Resulting Microword: 0248 FFFF FF\_E

Purpose: Clear Registers 1-B, D & E

Comments: Bits 35-32 specify that the function will be zero. Therefore, the register indicated by bits 7-4 will be cleared. The Am2910 is instructed to continue to the next address.

Mnemonic: LOOPBEG

Micro Routine: Bitpro  
Micro address: 0013

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#0	Operand sources from RAM
39-36	H#4	Destination to RAM
35-32	H#4	F=S+ carry in
31-30	B#01	Carry in = 1
29-24	Q#XX	Don't care
23	B#1	Don't latch micro
22	B#1	Don't latch macro
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Don't care
13-12	B#XX	Don't care
11-8	H#F	Ra=RF
7-4	H#F	Rb=RF
3-0	H#E	Continue

Resulting Microword: 0044 7FFF FFFE

Purpose: Update PC in register F

Comments: Bits 35-32 specify that the function will be equal to the value of the register F specified by bits 11-8 plus the carry in, which in this case is equal to one as specified by bits 31-30. The value is then stored in register F as indicated by bits 7-4. The Am2920 is instructed to continue to the next address.

Micro Routine: Bitpro  
Micro address: 0014

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#1	Disable Y output
42-40	Q#0	Operand Sources from RAM
39-36	H#4	Destination to RAM with parity
35-32	H#X	Don't care
31-30	B#XX	Don't care
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#01	Select command overlay
19-16	H#3	Read from memory
15	B#1	Don't set breakpoint
14	X	Spare/Don't care
13-12	B#XX	Don't care
11-8	H#F	Ra=RF
7-4	H#C	Rb=RC
3-0	H#E	Continue

Resulting Microword: 084F FFD3 FFCE

Purpose: Load counter from macro memory into register C

Comments: Bits 47-45 specify that the sources Ra and Rb are to be specified by the pipeline at bits 11-8 and 7-4 respectively. Since bits 19-16 specify the command to read from memory, then Ra=RF specifies a macro address and since RF is the program counter, the address specified is the next address in the program. With Rb=RC at bits 7-4, the destination for the value of the macro address is register C. The Am2910 is instructed to continue to the next sequential instruction.

# Mnemonic: STAGE1

Micro Routine: Bitpro

Micro address: 0015

BITS	VALUE	EXPLANATION
47-45	Q#4	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#0	Operand Sources from RAM
39-36	H#4	Destination to RAM
35-32	H#1	F=S-R-1 plus carry in
31-30	B#01	Carry in equal to one
29-24	Q#20	ALU status to status registers
23	B#0	Latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare/Don't care
13-12	B#XX	Don't care
11-8	H#2	Ra=R2
7-4	H#C	Rb=RC
3-0	H#E	Continue

Resulting Microword: 8041 507F F2CE

Purpose: R0=RC-R2

Comments: Bits 35-32 specify the function to be F=S-R-1 plus carry in. A carry in of one specified by bits 31-30 make the function F=S-R. S=RC and R=R2 as specified by the pipeline, bits 11-4 and the destination of the result is specified by bits 47-45 to be the register indicated by the instruction register. Since the Macro instruction is 01F0, the destination register is R0. The Am2910 is instructed to continue to the next sequential micro instruction.



Micro Routine: Bitpro  
Micro address: 0016

BITS	VALUE	EXPLANATION
47-45	Q#X	Don't care
44	B#X	Don't care
43	B#X	Don't care
42-40	Q#X	Don't care
39-36	H#X	Don't care
35-32	H#X	Don't care
31-30	B#X	Don't care
29-24	Q#X	Don't care
23	B#X	Don't care
22	B#X	Don't care
21-20	B#X	Don't care
19-16	H#X	Don't care
15	B#1	Don't set break point
14	X	Spare/don't care
13-12	B#10	This is now the
11-8	H#5	address field with
7-4	H#0	address H#250
3-0	H#C	Load address into R/C register

Resulting Microword: FFFF FFFF E50C

Purpose: Load R/C register with address H#250

Comments: The Am2910 is instructed to load the address specified in the pipeline into the R/C register and continue to the next address. The address loaded is for the shifting microroutine for the positive or zero case of the result of the previous adder stage instruction. The Am2910 also continues to the next sequential address.

Mnemonic: STAGE1

Micro Routine: Bitpro  
Micro address: 0017

BITS	VALUE	EXPLANATION
47-45	Q#X	Don't care
44	B#X	Don't care
43	B#X	Don't care
42-40	Q#X	Don't care
39-36	H#X	Don't care
35-32	H#X	Don't care
31-30	B#11	Use bit 29-24
29-24	Q#1F	Test if Micro negative
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#01	Command overlay
19-16	H#9	Enable true test
15	B#1	Don't set break point
14	X	Spare
13-12	B#10	This is the address
11-8	H#7	field with
7-4	H#0	address H#270
3-0	H#5	Conditional jump; True-pipeline address False-R/C address

Resulting Microword: FFFF DFD9 E705

Purpose: Conditional jump to address H#270 if micro status is negative (true), jump to address H#250 if not negative (false)

Comments: Bits 31-24 specify the Am2904 to test to see if the micro status bit is negative. The Am2910 then has a conditional jump on the results of this test to jump to the address in register R/C if false and to the address in the pipeline field if true.

Micro Routine: Bitpro  
Micro address: 0018

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#2	Operand From RAM
39-36	H#4	F to RAM
35-32	H#6	F=R + Carry in
31-30	B#00	Carry in equal to zero
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#1	Ra=R1
7-4	H#2	Rb=R2
3-0	H#E	Continue

Resulting Microword: 0246 3FFF F12E

Purpose: Place value of R1 into R2

Comments: Bits 35-30 specify the function to be F=R with carry in equal to zero. Bits 11-8 specify R to be R1 and bits 7-4 specify R2 to be the destination. The Am2910 is instructed to continue to the next sequential micro instruction.

Mnemonic: STAGE1

Micro Routine: Bitpro  
Micro address: 0019

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#2	Operand From RAM
39-36	H#4	F to RAM
35-32	H#6	F=R + Carry in
31-30	B#00	Carry in equal to zero
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#C	Ra=RC
7-4	H#1	Rb=R1
3-0	H#E	Continue

Resulting Microword: 0246 3FFF FC1E

Purpose: Place value of RC into R1

Comments: Bits 35-30 specify the function to be F=R with carry in equal to zero. Bits 11-8 specify R to be RC and bits 7-4 specify R1 to be the destination. The Am2910 is instructed to continue to the next sequential micro instruction.

Mnemonic: STAGE2

Micro Routine: Bitpro  
Micro address: 001A

BITS	VALUE	EXPLANATION
47-45	Q#2	Ra source & dest. from pipeline, Rb fm IR
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#0	Operand Sources from RAM
39-36	H#4	Destination to RAM
35-32	H#3	F=R + S plus carry in
31-30	B#00	Carry in equal to zero
29-24	Q#20	ALU status to status registers
23	B#0	Latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare/Don't care
13-12	B#XX	Don't care
11-8	H#6	Ra=R6
7-4	H#C	Rb=RC
3-0	H#E	Continue

Resulting Microword: 4043 107F F6CE

Purpose: RC=R0+R6

Comments: Bits 35-30 specify the function to be F=S+R with carry in equal to zero. S=R0 as specified by bits 47-45, and R=R6 and destination = RC as specified by bits 11-4. The Am2910 is instructed to continue to the next sequential micro instruction.

Micro Routine: Bitpro  
Micro address: 001B

BITS	VALUE	EXPLANATION
47-45	Q#X	Don't care
44	B#X	Don't care
43	B#X	Don't care
42-40	Q#X	Don't care
39-36	H#X	Don't care
35-32	H#X	Don't care
31-30	B#X	Don't care
29-24	Q#X	Don't care
23	B#X	Don't care
22	B#X	Don't care
21-20	B#X	Don't care
19-16	H#X	Don't care
15	B#1	Don't set break point
14	X	Spare/don't care
13-12	B#10	This is now the
11-8	H#1	address field with
7-4	H#0	address H#210
3-0	H#C	Load address into R/C register

Resulting Microword: FFFF FFFF E10C

Purpose: Load R/C register with address H#210

Comments: The Am2910 is instructed to load the address specified in the pipeline into the R/C register and continue to the next address. The address loaded is for the shifting microroutine for the positive or zero case of the result of the previous adder stage instruction. The Am2910 also continues to the next sequential address.

# Mnemonic: STAGE2

Micro Routine: Bitpro

Micro address: 001C

BITS	VALUE	EXPLANATION
47-45	Q#X	Don't care
44	B#X	Don't care
43	B#X	Don't care
42-40	Q#X	Don't care
39-36	H#X	Don't care
35-32	H#X	Don't care
31-30	B#11	Use bit 29-24
29-24	Q#1F	Test if Micro negative
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#01	Command overlay
19-16	H#9	Enable true test
15	B#1	Don't set break point
14	X	Spare
13-12	B#10	This is the address
11-8	H#3	field with
7-4	H#0	address H#230
3-0	H#5	Conditional jump; True-pipeline address False-R/C address

Resulting Microword: FFFF DFD9 E305

Purpose: Conditional jump to address H#230 if micro status is negative (true), jump to address H#210 if not negative (false)

Comments: Bits 31-24 specify the Am2904 to test to see if the micro status bit is negative. The Am2910 then has a conditional jump on the results of this test to jump to the address in register R/C if false and to the address in the pipeline field if true.

Micro Routine: Bitpro  
Micro address: 001D

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#2	Operand From RAM
39-36	H#4	F to RAM
35-32	H#6	F=R + Carry in
31-30	B#00	Carry in equal to zero
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#5	Ra=R5
7-4	H#6	Rb=R6
3-0	H#E	Continue

Resulting Microword: 0246 3FFF F56E

Purpose: Place value of R5 into R6

Comments: Bits 35-30 specify the function to be F=R with carry in equal to zero. Bits 11-8 specify R to be R5 and bits 7-4 specify R6 to be the destination. The Am2910 is instructed to continue to the next sequential micro instruction.



Mnemonic: STAGE2

Micro Routine: Bitpro  
Micro address: 001E

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#2	Operand From RAM
39-36	H#4	F to RAM
35-32	H#6	F=R + Carry in
31-30	B#00	Carry in equal to zero
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#4	Ra=R4
7-4	H#5	Rb=R5
3-0	H#E	Continue

Resulting Microword: 0246 3FFF F45E

Purpose: Place value of R4 into R5

Comments: Bits 35-30 specify the function to be F=R with carry in equal to zero. Bits 11-8 specify R to be R4 and bits 7-4 specify R5 to be the destination. The Am2910 is instructed to continue to the next sequential micro instruction.

Micro Routine: Bitpro  
Micro address: 001F

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#2	Operand From RAM
39-36	H#4	F to RAM
35-32	H#6	F=R + Carry in
31-30	B#00	Carry in equal to zero
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#3	Ra=R3
7-4	H#4	Rb=R4
3-0	H#E	Continue

Resulting Microword: 0246 3FFF F34E

Purpose: Place value of R3 into R4

Comments: Bits 35-30 specify the function to be F=R with carry in equal to zero. Bits 11-8 specify R to be R3 and bits 7-4 specify R4 to be the destination. The Am2910 is instructed to continue to the next sequential micro instruction.

Mnemonic: STAGE2

Micro Routine: Bitpro  
Micro address: 0020

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#2	Operand From RAM
39-36	H#4	F to RAM
35-32	H#6	F=R + Carry in
31-30	B#00	Carry in equal to zero
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#0	Ra=R0
7-4	H#3	Rb=R3
3-0	H#E	Continue

Resulting Microword: 0246 3FFF F03E

Purpose: Place value of R0 into R3

Comments: Bits 35-30 specify the function to be F=R with carry in equal to zero. Bits 11-8 specify R to be R0 and bits 7-4 specify R3 to be the destination. The Am2910 is instructed to continue to the next sequential micro instruction.

Mnemonic: STAGE3

Micro Routine: Bitpro  
Micro address: 0021

BITS	VALUE	EXPLANATION
47-45	Q#4	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#0	Operand Sources from RAM
39-36	H#4	Destination to RAM
35-32	H#3	F=R + S plus carry in
31-30	B#01	Carry in equal to zero
29-24	Q#20	ALU status to status registers
23	B#0	Latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare/Don't care
13-12	B#XX	Don't care
11-8	H#9	Ra=R9
7-4	H#C	Rb=RC
3-0	H#E	Continue

Resulting Microword: 8043 107F F9CE

Purpose: R0=RC+R9

Comments: Bits 35-30 specify the function to be  $F=S+R$  with carry in equal to zero.  $S=RC$  and  $R=R9$  as specified by the pipeline, bits 11-4 and the destination of the result is specified by bits 47-45 to be the register indicated by the instruction register. Since the Macro instruction is 01F0, the destination register is R0. The Am2910 is instructed to continue to the next sequential micro instruction.

Micro Routine: Bitpro  
Micro address: 0022

BITS	VALUE	EXPLANATION
47-45	Q#X	Don't care
44	B#X	Don't care
43	B#X	Don't care
42-40	Q#X	Don't care
39-36	H#X	Don't care
35-32	H#X	Don't care
31-30	B#X	Don't care
29-24	Q#X	Don't care
23	B#X	Don't care
22	B#X	Don't care
21-20	B#X	Don't care
19-16	H#X	Don't care
15	B#1	Don't set break point
14	X	Spare/don't care
13-12	B#10	This is now the
11-8	H#5	address field with
7-4	H#0	address H#250
3-0	H#C	Load address into R/C register

Resulting Microword: FFFF FFFF E50C

Purpose: Load R/C register with address H#250

Comments: The Am2910 is instructed to load the address specified in the pipeline into the R/C register and continue to the next address. The address loaded is for the shifting microroutine for the positive or zero case of the result of the previous adder stage instruction. The Am2910 also continues to the next sequential address.

Mnemonic: STAGE3

Micro Routine: Bitpro  
Micro address: 0023

BITS	VALUE	EXPLANATION
-----		
47-45	Q#X	Don't care
44	B#X	Don't care
43	B#X	Don't care
42-40	Q#X	Don't care
39-36	H#X	Don't care
35-32	H#X	Don't care
31-30	B#11	Use bit 29-24
29-24	Q#1F	Test if Micro negative
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#01	Command overlay
19-16	H#9	Enable true test
15	B#1	Don't set break point
14	X	Spare
13-12	B#10	This is the address
11-8	H#7	field with
7-4	H#0	address H#270
3-0	H#5	Conditional jump; True-pipeline address False-R/C address

Resulting Microword: FFFF DFD9 E705

Purpose: Conditional jump to address H#270 if micro status is negative (true), jump to address H#250 if not negative (false)

Comments: Bits 31-24 specify the Am2904 to test to see if the micro status bit is negative. The Am2910 then has a conditional jump on the results of this test to jump to the address in register R/C if false and to the address in the pipeline field if true.

Micro Routine: Bitpro  
Micro address: 0024

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#2	Operand From RAM
39-36	H#4	F to RAM
35-32	H#6	F=R + Carry in
31-30	B#00	Carry in equal to zero
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#1	Ra=R8
7-4	H#2	Rb=R9
3-0	H#E	Continue

Resulting Microword: 0246 3FFF F89E

Purpose: Place value of R8 into R9

Comments: Bits 35-30 specify the function to be F=R with carry in equal to zero. Bits 11-8 specify R to be R8 and bits 7-4 specify R9 to be the destination. The Am2910 is instructed to continue to the next sequential micro instruction.

Mnemonic: STAGE3

Micro Routine: Bitpro  
Micro address: 0025

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#2	Operand From RAM
39-36	H#4	F to RAM
35-32	H#6	F=R + Carry in
31-30	B#00	Carry in equal to zero
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#7	Ra=R7
7-4	H#8	Rb=R8
3-0	H#E	Continue

Resulting Microword: 0246 3FFF F78E

Purpose: Place value of R7 into R8

Comments: Bits 35-30 specify the function to be F=R with carry in equal to zero. Bits 11-8 specify R to be R7 and bits 7-4 specify R8 to be the destination. The Am2910 is instructed to continue to the next sequential micro instruction.



Micro Routine: Bitpro  
Micro address: 0026

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#2	Operand From RAM
39-36	H#4	F to RAM
35-32	H#6	F=R + Carry in
31-30	B#00	Carry in equal to zero
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#C	Ra=RC
7-4	H#7	Rb=R7
3-0	H#E	Continue

Resulting Microword: 0246 3FFF FC7E

Purpose: Place value of RC into R7

Comments: Bits 35-30 specify the function to be F=R with carry in equal to zero. Bits 11-8 specify R to be RC and bits 7-4 specify R7 to be the destination. The Am2910 is instructed to continue to the next sequential micro instruction.

Mnemonic: STAGE4

Micro Routine: Bitpro  
Micro address: 0027

BITS	VALUE	EXPLANATION
47-45	Q#2	Ra source & dest. from pipeline, Rb fm IR
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#0	Operand Sources from RAM
39-36	H#4	Destination to RAM
35-32	H#1	F=S-R-1 plus carry in
31-30	B#01	Carry in equal to one
29-24	Q#20	ALU status to status registers
23	B#0	Latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare/Don't care
13-12	B#XX	Don't care
11-8	H#A	Ra=RA
7-4	H#C	Rb=RC
3-0	H#E	Continue

Resulting Microword: 4041 507F FACE

Purpose: RC=R0-RA

Comments: Bits 35-32 specify the function to be F=S-R-1 plus carry in. A carry in of one specified by bits 31-30 make the function F=S-R. S=R0 as specified by bits 47-45, and R=RA and destination = RC as specified by bits 11-4. The Am2910 is instructed to continue to the next sequential micro instruction.

Micro Routine: Bitpro  
Micro address: 0028

BITS	VALUE	EXPLANATION
47-45	Q#X	Don't care
44	B#X	Don't care
43	B#X	Don't care
42-40	Q#X	Don't care
39-36	H#X	Don't care
35-32	H#X	Don't care
31-30	B#X	Don't care
29-24	Q#X	Don't care
23	B#X	Don't care
22	B#X	Don't care
21-20	B#X	Don't care
19-16	H#X	Don't care
15	B#1	Don't set break point
14	X	Spare/don't care
13-12	B#10	This is now the
11-8	H#1	address field with
7-4	H#0	address H#210
3-0	H#C	Load address into R/C register

Resulting Microword: FFFF FFFF E10C

Purpose: Load R/C register with address H#210

Comments: The Am2910 is instructed to load the address specified in the pipeline into the R/C register and continue to the next address. The address loaded is for the shifting microroutine for the positive or zero case of the result of the previous adder stage instruction. The Am2910 also continues to the next sequential address.

Mnemonic: STAGE4

Micro Routine: Bitpro  
Micro address: 0029

BITS	VALUE	EXPLANATION
47-45	Q#X	Don't care
44	B#X	Don't care
43	B#X	Don't care
42-40	Q#X	Don't care
39-36	H#X	Don't care
35-32	H#X	Don't care
31-30	B#11	Use bit 29-24
29-24	Q#1F	Test if Micro negative
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#01	Command overlay
19-16	H#9	Enable true test
15	B#1	Don't set break point
14	X	Spare
13-12	B#10	This is the address
11-8	H#3	field with
7-4	H#0	address H#230
3-0	H#5	Conditional jump; True-pipeline address False-R/C address

Resulting Microword: FFFF DFD9 E305

Purpose: Conditional jump to address H#230 if micro status is negative (true), jump to address H#210 if not negative (false)

Comments: Bits 31-24 specify the Am2904 to test to see if the micro status bit is negative. The Am2910 then has a conditional jump on the results of this test to jump to the address in register R/C if false and to the address in the pipeline field if true.

Micro Routine: Bitpro  
Micro address: 002A

BITS	VALUE	EXPLANATION
-----		
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#2	Operand From RAM
39-36	H#4	F to RAM
35-32	H#6	F=R + Carry in
31-30	B#00	Carry in equal to zero
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#0	Ra=R0
7-4	H#A	Rb=RA
3-0	H#E	Continue

Resulting Microword: 0246 3FFF FOAE

Purpose: Place value of R0 into RA

Comments: Bits 35-30 specify the function to be F=R with carry in equal to zero. Bits 11-8 specify R to be R0 and bits 7-4 specify RA to be the destination. The Am2910 is instructed to continue to the next sequential micro instruction.

Mnemonic: STAGE5

Micro Routine: Bitpro  
Micro address: 002B

BITS	VALUE	EXPLANATION
47-45	Q#4	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#0	Operand Sources from RAM
39-36	H#4	Destination to RAM
35-32	H#1	F=S-R-1 plus carry in
31-30	B#01	Carry in equal to one
29-24	Q#20	ALU status to status registers
23	B#0	Latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare/Don't care
13-12	B#XX	Don't care
11-8	H#B	Ra=RB
7-4	H#C	Rb=RC
3-0	H#E	Continue

Resulting Microword: 8041 507F FBCE

Purpose: R0=RC-RB

Comments: Bits 35-32 specify the function to be  $F=S-R-1$  plus carry in. A carry in of one specified by bits 31-30 make the function  $F=S-R$ .  $S=RC$  and  $R=RB$  as specified by the pipeline, bits 11-4 and the destination of the result is specified by bits 47-45 to be the register indicated by the instruction register. Since the Macro instruction is 01F0, the destination register is R0. The Am2910 is instructed to continue to the next sequential micro instruction.

Micro Routine: Bitpro  
Micro address: 002C

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#2	Operand From RAM
39-36	H#4	F to RAM
35-32	H#6	F=R + Carry in
31-30	B#00	Carry in equal to zero
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#C	Ra=RC
7-4	H#B	Rb=RB
3-0	H#E	Continue

Resulting Microword: 0246 3FFF FCBE

Purpose: Place value of RC into RB

Comments: Bits 35-30 specify the function to be F=R with carry in equal to zero. Bits 11-8 specify R to be RC and bits 7-4 specify RB to be the destination. The Am2910 is instructed to continue to the next sequential micro instruction.

Mnemonic: STAGE6

Micro Routine: Bitpro  
Micro address: 002D

BITS	VALUE	EXPLANATION
47-45	Q#2	Ra source & dest. from pipeline, Rb fm IR
44	B#Q	Enable Am29203
43	B#0	Enable Y output
42-40	Q#0	Operand Sources from RAM
39-36	H#4	Destination to RAM
35-32	H#1	F=S-R-1 plus carry in
31-30	B#01	Carry in equal to one
29-24	Q#20	ALU status to status registers
23	B#0	Latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare/Don't care
13-12	B#XX	Don't care
11-8	H#E	Ra=RE
7-4	H#C	Rb=RC
3-0	H#E	Continue

Resulting Microword: 4041 507F FECE

Purpose: RC=R0-RE

Comments: Bits 35-32 specify the function to be F=S-R-1 plus carry in. A carry in of one specified by bits 31-30 make the function F=S-R. S=R0 as specified by bits 47-45, and R=RE and destination = RC as specified by bits 11-4. The Am2910 is instructed to continue to the next sequential micro instruction.



Micro Routine: Bitpro  
Micro address: 002E

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#1	Disable 29203
43	B#0	Enable Y output
42-40	Q#0	Operand Sources from RAM
39-36	H#C	F to Y only
35-32	H#4	F=S plus carry in
31-30	B#00	Carry in equal to zero
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#01	Command overlay
19-16	H#4	Write to memory
15	B#1	Don't set break point
14	X	Spare
13-12	B#XX	Don't care
11-8	H#F	Ra=RF
7-4	H#C	Rb=RC
3-0	H#E	Continue

Resulting Microword: 10C4 3FD4 FFCE

Purpose: Place result of filter stored in register C into macro memory address pointed to by the PC

Comments: The command field of the Am2904, bits 21-16, specifies to write to memory. It writes to the location pointed to by Ra which in this case is RF, the PC. It places the value from RC into this memory location. The Am2910 is instructed to continue to the next sequential address.

Mnemonic: STAGE6

Micro Routine: Bitpro  
Micro address: 002F

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#2	Operand From RAM
39-36	H#4	F to RAM
35-32	H#6	F=R + Carry in
31-30	B#00	Carry in equal to zero
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#D	Ra=RD
7-4	H#E	Rb=RE
3-0	H#E	Continue

Resulting Microword: 0246 3FFF FDEE

Purpose: Place value of RD into RE

Comments: Bits 35-30 specify the function to be F=R with carry in equal to zero. Bits 11-8 specify R to be RD and bits 7-4 specify RE to be the destination. The Am2910 is instructed to continue to the next sequential micro instruction.

Micro Routine: Bitpro  
Micro address: 0030

BITS	VALUE	EXPLANATION
-----		
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#2	Operand From RAM
39-36	H#4	F to RAM
35-32	H#6	F=R + Carry in
31-30	B#00	Carry in equal to zero
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#0	Ra=R0
7-4	H#D	Rb=RD
3-0	H#E	Continue

Resulting Microword: 0246 3FFF FODE

Purpose: Place value of R0 into RD

Comments: Bits 35-30 specify the function to be F=R with carry in equal to zero. Bits 11-8 specify R to be R0 and bits 7-4 specify RD to be the destination. The Am2910 is instructed to continue to the next sequential micro instruction.

Mnemonic: DECCNTR

Micro Routine: Bitpro  
Micro address: 0031

BITS	VALUE	EXPLANATION
47-45	Q#4	Sources Ra & Rb from pipeline, Dest fm IR
44	B#0	Enable Am29203
43	B#0	Enabel Y output
42-40	Q#2	Operand S from Q register
39-36	H#4	Destination to RAM
35-32	H#4	F=S plus carry in
31-30	B#00	Carry in equal to zero
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#X	Don't care
7-4	H#X	Don't care
3-0	H#E	Continue

Resulting Microword: 8244 3FFF FFFE

Purpose: Put counter from Q register into register 0

Comments: The operand S comes from the Q register as specified by bits 42-40 and is placed in register R0 since bits 47-45 specify the destination to be indicated by the IR and the macro instruction in this case is 01F0. The Am2910 is instructed to continue to the next sequential address.

Micro Routine: Bitpro  
Micro address: 0032

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#0	Operand Sources from RAM
39-36	H#3	SPECIAL FUNCTION: Decrement by 1
35-32	H#0	ALU special function
31-30	B#01	One to be decremented (00 would decr 2)
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#X	Don't care
7-4	H#0	Rb=R0
3-0	H#E	Continue

Resulting Microword: 0030 7FFF FF0E

Purpose: Decrement counter by one

Comments: This instruction is an ALU special function as designated by bits 35-32. Bits 39-36 specify the special function to be a decrement and since bits 31-30 are 01, the decrement is to be one. The operand is R0 as specified by bits 7-4. The Am2910 is instructed to continue to the next sequential address.

Mnemonic: DECCNTR

Micro Routine: Bitpro  
Micro address: 0033

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb specified by pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#0	Operand sources from RAM
39-36	H#6	F to Q register
35-32	H#4	F=S plus carry in
31-30	B#00	Carry in equal to zero
29-24	Q#20	ALU status to status registers
23	B#0	Latch micro status
22	B#1	Latch macro status
21-20	B#11	No command or shift
19-16	H#F	Don't care
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#X	Don't care
7-4	H#0	Rb=R0
3-0	H#E	Continue

Resulting Microword: 0064 107F FF0E

Purpose: Load counter back into Q register and check if counter is zero

Comments: Bits 39-36 specify the result destination to the Q register. Bits 35-30 specify the function to be F=S with carry in equal to zero and S is designated to be R0 as specified by bits 7-4. The Am2910 is instructed to continue to the next sequential address.

Micro Routine: Bitpro  
Micro address: 0034

BITS	VALUE	EXPLANATION
47-45	Q#X	Don't care
44	B#X	Don't care
43	B#X	Don't care
42-40	Q#X	Don't care
39-36	H#X	Don't care
35-32	H#X	Don't care
31-30	B#11	Use bits 29-24
29-24	Q#14	Test: Micro not zero
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#01	Command overlay
19-16	H#9	Allow true test
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#00	This is now the
11-8	H#1	address field with
7-4	H#3	address H#013
3-0	H#3	Cond. jump to pipeline address if true

Resulting Microword: FFFF D4D9 C133

Purpose: Jump back to beginning of filter to load new data point, if counter is not zero

Comments: Bits 31-24 test the zero status bit to see if it is not zero. If this test is true, the Am2910 jumps to the address H#013 as specified by bits 3-0 and bits 13-4 respectively. Otherwise, the Am2910 would continue to the next sequential address which would most likely be a branch to the next instruction fetch.

Micro Routine: POSSHFTC  
Micro address: 0210

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb from pipeline
44	B#1	Disable Am29203
43	B#0	Enable Y output
42-40	Q#0	Both Sources from RAM
39-36	H#4	Destination to RAM
35-32	H#4	R=S + carry in
31-30	B#00	Carry in equal to zero
29-24	Q#20	ALU status to status registers
23	B#0	Latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#X	Don't care
7-4	H#C	Rb=RC
3-0	H#E	Continue

Resulting Microword: 1044 107F FFCE

Purpose: Latch incoming data to test for zero

Comments: The purpose of this instruction is merely to test the data point in RC for zero and load the micro status registers with the result. Bits 29-23 specify the ALU status to be loaded and bits 7-4 designate RC to be tested.



Micro Routine: POSSHFTC  
Micro address: 0211

BITS	VALUE	EXPLANATION
47-45	Q#X	Don't care
44	B#X	Don't care
43	B#X	Don't care
42-40	Q#X	Don't care
39-36	H#X	Don't care
35-32	H#X	Don't care
31-30	B#11	Use bits 29-24
29-24	Q#15	Test: Micro Zero
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#01	Command overlay
19-16	H#9	Allow true status register test
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#10	This is the
11-8	H#2	address field for
7-4	H#3	address H#223
3-0	H#3	Jump to pipeline address if test true

Resulting Microword: FFFF D5D5 E233

Purpose: Test for zero, if true - go to return,  
if false - continue

Comments: This Am2904 command, bits 19-16, orders a true test of the status registers for zero. If true, the Am2910 instruction jumps to the pipeline address. If false, the Am2910 continues to the next sequential address.

Micro Routine: POSSSHFTC  
Micro address: 0212-021A

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb from pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#0	RAM source for operands
39-36	H#0	F to RAM, arithmetic down shift
35-32	H#4	F=S plus carry in
31-30	B#00	Carry in equal to zero
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#10	Shift overlay
19-16	H#0	Shift right, zero fill
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#X	Don't care
7-4	H#C	Rb=RC
3-0	H#E	Continue

Resulting Microword: 0004 3FE0 FFCE

Purpose: Shift zero into MSB, shift out LSB

Comments: RC is the source and destination for the shift.  
Bits 39-36 specify the shift to be downshift and bits 21-16  
specify the shift to be zero fill. The Am2910 continues to  
the next sequential address.

Micro Routine: POSSHFTC  
Micro address: 021B-0222

BITS	VALUE	EXPLANATION
-----		
47-45	Q#0	Sources Ra & Rb from pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#0	RAM source for operands
39-36	H#8	F to RAM, arithmetic upshift
35-32	H#4	F=S plus carry in
31-30	B#00	Carry in equal to zero
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#10	Shift overlay
19-16	H#0	Shift left, zero fill
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#X	Don't care
7-4	H#C	Rb=RC
3-0	H#E	Continue

Resulting Microword: 0084 3FE0 FFCE

Purpose: Shift zero into LSB, shift out MSB

Comments: RC is the source and destination for the shift. Bits 39-36 specify the shift to be upshift and bits 21-16 specify the shift to be zero fill. The Am2910 continues to the next sequential address.

Micro Routine: POSSSHFTC  
Micro address: 0223

BITS	VALUE	EXPLANATION
------	-------	-------------

47-45	Q#X	Don't care
44	B#X	Don't care
43	B#X	Don't care
42-40	Q#X	Don't care
39-36	H#X	Don't care
35-32	H#X	Don't care
31-30	B#XX	Don't care
29-24	Q#XX	Don't care
23	B#X	Don't care
22	B#X	Don't care
21-20	B#XX	Don't care
19-16	H#9	Forced pass
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#X	Don't care
7-4	H#X	Don't care
3-0	H#A	Conditional return

Resulting Microword: FFFF FFF9 FFFA

Purpose: Return to calling microroutine

Comments: With the forced pass on the conditional return, the Am2910 returns to the address on the stack which is back to the calling microroutine.

Micro Routine: NEGSHTC  
Micro address: 0230-0238

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb from pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#0	RAM source for operands
39-36	H#0	F to RAM, arithmetic down shift
35-32	H#4	F=S plus carry in
31-30	B#00	Carry in equal to zero
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#10	Shift overlay
19-16	H#1	Shift right, one fill
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#X	Don't care
7-4	H#C	Rb=RC
3-0	H#E	Continue

Resulting Microword: 0004 3FE1 FFCE

Purpose: Shift one into MSB, shift out LSB

Comments: RC is the source and destination for the shift.  
Bits 39-36 specify the shift to be downshift and bits 21-16  
specify the shift to be one fill. The Am2910 continues to  
the next sequential address.

Micro Routine: NEGSHTC  
Micro address: 0239

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb from pipeline
44	B#1	Disable Am29203
43	B#0	Enable Y output
42-40	Q#0	Both Sources from RAM
39-36	H#4	Destination to RAM
35-32	H#4	R=S + carry in
31-30	B#00	Carry in equal to zero
29-24	Q#20	ALU status to status registers
23	B#0	Latch micro status
22	B#1	Don't latch macro status
21-20	B#11	No command or shift
19-16	H#X	Don't care
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#X	Don't care
7-4	H#C	Rb=RC
3-0	H#E	Continue

Resulting Microword: 1044 107F FFCE

Purpose: Latch data to test for zero

Comments: The purpose of this instruction is merely to test the data point in RC for zero and load the micro status registers with the result. Bits 29-23 specify the ALU status to be loaded and bits 7-4 designate RC to be tested. The Am2910 continues to the next sequential address.

Micro Routine: NEGSHTC  
Micro address: 023A

BITS	VALUE	EXPLANATION
47-45	Q#X	Don't care
44	B#X	Don't care
43	B#X	Don't care
42-40	Q#X	Don't care
39-36	H#X	Don't care
35-32	H#X	Don't care
31-30	B#11	Use bits 29-24
29-24	Q#15	Test:  Micro Zero
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#01	Command overlay
19-16	H#9	Allow true status register test
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#10	This is the
11-8	H#4	address field for
7-4	H#3	address H#243
3-0	H#3	Jump to pipeline address if test true

Resulting Microword: FFFF D5D5 E433

Purpose: Test for zero, if true - go to return,  
          if false - continue

Comments: This Am2904 command, bits 19-16, orders a true test of the status registers for zero. If true, the Am2910 instruction jumps to the pipeline address. If false, the Am2910 continues to the next sequential address.

Micro Routine: NEGSHTC  
Micro address: 023B-0242

BITS	VALUE	EXPLANATION
47-45	Q#0	Sources Ra & Rb from pipeline
44	B#0	Enable Am29203
43	B#0	Enable Y output
42-40	Q#0	RAM source for operands
39-36	H#8	F to RAM, arithmetic upshift
35-32	H#4	F=S plus carry in
31-30	B#00	Carry in equal to zero
29-24	Q#XX	Don't care
23	B#1	Don't latch micro status
22	B#1	Don't latch macro status
21-20	B#10	Shift overlay
19-16	H#0	Shift left, zero fill
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#X	Don't care
7-4	H#C	Rb=RC
3-0	H#E	Continue

Resulting Microword: 0084 3FE0 FFCE

Purpose: Shift zero into LSB, shift out MSB

Comments: RC is the source and destination for the shift.  
Bits 39-36 specify the shift to be upshift and bits 21-16  
specify the shift to be zero fill. The Am2910 continues to  
the next sequential address.



Micro Routine: NEGSHTFC  
Micro address: 0243

BITS	VALUE	EXPLANATION
-----		
47-45	Q#X	Don't care
44	B#X	Don't care
43	B#X	Don't care
42-40	Q#X	Don't care
39-36	H#X	Don't care
35-32	H#X	Don't care
31-30	B#XX	Don't care
29-24	Q#XX	Don't care
23	B#X	Don't care
22	B#X	Don't care
21-20	B#XX	Don't care
19-16	H#9	Forced pass
15	B#1	Don't set breakpoint
14	X	Spare
13-12	B#XX	Don't care
11-8	H#X	Don't care
7-4	H#X	Don't care
3-0	H#A	Conditional return

Resulting Microword: FFFF FFF9 FFFA

Purpose: Return to calling microroutine

Comments: With the forced pass on the conditional return,  
the Am2910 returns to the address on the stack which is back  
to the calling microroutine.

Micro Routine: POSSHFT0  
Micro Address: 0250-263

This routine is identical to microroutine POSSHFTC with the following exceptions:

Replace register C with register 0 in all microcode

Replace address in pipeline with address H#263

---

Micro Routine: NEGSHT0  
Micro Routine: 0270-0283

This routine is identical to microroutine NEGSHTC with the following exceptions:

Replace register C with register 0 in all microcode

Replace address in pipeline with address H#283

# APPENDIX C

## FORTRAN PROGRAM OF FIR FILTER WITH CPU TIMING ROUTINE ADDED

```

C
C      THIS PROGRAM IS A REPRESENTATION OF A 13TH ORDER BAND PASS FILTER
C
      INTEGER X(200),Y(200)
      REAL *8 T(201)
      INTEGER N
      HANDLE=0
      PRINT 4
4      FORMAT ('1')
      CALL INPUT (N,X,T)
      CALL FUNCT (X,Y,N)
      PRINT 4
      STOP
      END

C -----
-
      SUBROUTINE INPUT (N,X,TIME1)
      REAL *8 XX(200),F,FS,TIME1(201),THETA
      INTEGER X(200)
      INTEGER N,K
      N=195
      F=3.58E6
      FS=1.42E7
      TIME1(1)=0.
      DO 100 K=1,N
          THETA=2.*3.1415926*F*TIME1(K)
          XX(K)=63.*SIN(THETA)
          X(K)=INT(XX(K))
          TIME1(K+1)=K/FS
100      CONTINUE
      RETURN
      END

C -----
      SUBROUTINE FUNCT (X,Y,N)
      REAL *8 TIMER1,TIMER2
      INTEGER X(200),Y(200),Y1,Y2,Y3,Y4,Y5
      INTEGER X1/O/,X2/O/,Y14/O/,Y13/O/,Y12/O/,Y11/O/,Y23/O/
      INTEGER Y22/O/,Y21/O/,Y31/O/,Y41/O/,Y52/O/,Y51/O/
      INTEGER K,N
      RMS=0.
      CALL JCPUT(TIMER1)
      DO 50 K=1,N
          Y1=X(K)-X2
          Y1=Y1/2
          X2=X1
          X1=X(K)
          Y2=Y1+Y14
          Y2=Y2/2
          Y14=Y13
          Y13=Y12
          Y12=Y11
          Y11=Y1

```

```

        Y3=Y2+Y23
        Y3=Y3/2
        Y23=Y22
        Y22=Y21
        Y21=Y2
        Y4=Y3-Y31
        Y4=Y4/2
        Y31=Y3
        Y5=Y4-Y41
        Y41=Y4
        Y(K)=Y5-Y52
        Y52=Y51
        Y51=Y5
    IF (K.GT.13) THEN
        RMS=RMS+Y(K)*Y(K)
    END IF
50    CONTINUE
    CALL JCPUT(TIMER2)
    TOTAL=TIMER2-TIMER1
    WRITE (13,271) TIMER1,TIMER2,TOTAL
271    FORMAT ( ' TIMER1 = ',D17.10,'TIMER2 = ',D17.10,'TOTAL = ',D17.10)
    RETURN
    END
C -----
    SUBROUTINE OUTPUT (X,Y,T,N)
    REAL *8 T(201)
    INTEGER X(200),Y(200),IHEX(200)
    INTEGER I,N
    DO 200 I=1,N
        IF (Y(I).LT.0) THEN
            IHEX(I)=Y(I)+256
        ELSE
            IHEX(I)=Y(I)
        END IF
        WRITE (13,201) I,IHEX(I),I,Y(I)
201    FORMAT ( ' HY',I3,' = ',Z2,5X,'Y',I3,' = ',I3)
200    CONTINUE
    RETURN
    END
C -----
    SUBROUTINE JCPUT(XCPUT)
C
C    RETURN CPU TIME AS A FLOATING PT VALUE
C
    PARAMETER JPI$_CPUTIME = '407'X
    INTEGER*2 BUF(8)
    INTEGER*4 BUF1(4),CPUT
    INTEGER SYS$GETJPI
    EQUIVALENCE(BUF(1),BUF1(1))
    REAL SCPUT
    BUF(1)=4
    BUF(2)=JPI$_CPUTIME
    BUF1(2)=%LOC(CPUT)
    BUF1(3)=0
    BUF(4)=0
    IRET=SYS$GETJPI(...,BUF,...)
    XCPUT=FLOAT(CPUT)/100.0
    RETURN
    END

```

### LIST OF REFERENCES

1. "Bit-Slice ICs Kick Off Era of Commercial GaAs LSI," Electronics, pp. 83-86, September 18, 1986.
2. Fischer, T., "Digital VLSI Breeds Next-generation TV Receivers," Electronics, pp. 97-103, August 11, 1981.
3. Hockney, R.W., Jesshope, C.R., Parallel Computers, pp. 146-153, Adam Helger Ltd, Bristol, 1981.
4. Adams, W.T., Smith, S.M., "How Bit-Slice Families Compare: Part 1, Evaluating Processor Elements," Electronics, pp. 91-98, August 3, 1978.
5. White, D.E., Bit-Slice Design: Controllers and ALUs, pp. 9, 30-42, 70-71, Garland STPM Press, 1981.
6. Wolfe, C.F., "Bit-slice Processors Come To Mainframe Design," Electronics, pp. 118-123, February 28, 1980.
7. Hartrum, T.C., and others, Am29203 Evaluation Board User's Guide, Advanced Micro Devices, Inc., 1986.
8. RM-9400 Series Graphic Display System Hardware Reference Manual, Publication Number 504616, Revision B, Volume 1, Ramtek Corporation Technical Publications, 1980.
9. Liskear, J., "The Bit-Slice Alternative (Graphics)," Computer Design, p. 44, January 15, 1985.
10. "Bipolar 8-bit Slice Family Includes PLAs," Computer Design, p. 105, December 15, 1985.
11. Lobo, K., and others, "Structured Arrays for Microprogrammed Systems," Semicustom Design Guide, pp. 44-53, Summer 1986.
12. Chen, C. T., One-Dimensional Digital Signal Processing, pp. 8-10, 191, Marcel Dekker, Inc., 1979.
13. Programming In VAX Fortran, V. AA-D034D-7E, pp. 3.3, 3.12, Digital Equipment Corporation, 1984.
14. Gold, B., Rabiner, L.R., Theory and Application of Digital Signal Processing, pp. 295-309, 337-349, Prentice-Hall, Inc., 1975.

15. Stewart, M.B., The Application of Bit Slice Design To Digital Image Processing, Masters Thesis, Naval Postgraduate School, Monterey, California, September 1986.
16. Becker, T.F., GaAs Microprocessors and Memories for High Speed System Design, Vitesse Electronics Corporation, 1986.
17. Smartcom II for IBM PC, IBM XT and Compatibles, Hayes Microcomputer Products, Inc., 1984.

## BIBLIOGRAPHY

- Adams, W.T., Smith, S.M., "How Bit-Slice Families Compare: Part 2, Sizing Up the Microcontroller," Electronics, August 17, 1978.
- Baker, S., "Microslice Family is a Logical Move," Electronics Weekly, November 13, 1985.
- Brick, J., Mick J., Bit-Slice Microprocessor Design, McGraw-Hill Book Company, 1980.
- DeMonrico, C., Laczko, F., "When Bit-Slices Team Up With ECL, 32-Bit Computers Rise to Superpower Status," Electronic Design, May 15, 1986.
- Everett, D., Thorpe, R., "Single Chip Combines Bit-Slice and EPROM," Computer Design, August 15, 1986.
- Freund, M., Kital, R., "Digital Distance Relay mho Elements Using Bit-Slice Technology," IEEE Transactions on Instrumentation and Measurement, Vol. IM-34, No. 4, December 1985.
- Kirk, D.E., Strum, R.D., First Principles of Discrete Systems and Digital Signal Processing, Addison-Wesley Publishing Co., 1987.
- Langdon, G.G., Computer Design, Computeach Press Inc., 1982.
- Stone, H.S., Microcomputer Interfacing, Addison-Wesley Publishing Co., 1982.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center Cameron Station Alexandrea, Virginia 22304-6145	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3. Department Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
4. Professor Chin-Hwa Lee, Code 62Le Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	4
5. Professor Mitchell Cotton, Code 62Cc Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
6. Commander Naval Surface Force U.S. Atlantic Fleet Norfolk, Virginia 23511-6292 Attention: Lieutenant Darrel W. Purdy	6



END

11-87

DTIC